ARIADNE USER GUIDE

Authors: Rafik CHIBOUT, Françoise SAILHAN, Valérie ISSARNY
{rafik.chibout, francoise.sailhan, valerie.issarny}@inria.fr

Table of contents

List of Figures

I Introduction

Mobile Ad hoc NETworks (MANETs) enable spontaneous connectivity between wireless devices, which can share their services, without requiring any particular infrastructure. In addition, ad hoc routing protocols increase connectivity by offering multi-hop communication. However, distributed service provision over MANETs requires adequate support of service discovery and invocation, due to the network's dynamics and the resource constraints of wireless devices. Some service discovery protocols for MANETs have been proposed, but they are mainly aimed at small-scale MANETs. We thus developed Ariadne [1], which is a scalable protocol for MANETs, based on the homogeneous and dynamic deployment of cooperating directories within the network. The scalability of Ariadne is made possible by minimizing the amount generated traffic and using of compact directory summaries. These summaries enable the directory that most likely caches the description of a given service, to be efficiently located. Ariadne is based on the Web Services architecture to provide interoperability among mobile devices. Specifically, we use WSDL (Web Service Description Language) [2] for service description and SOAP (Simple Object Access Protocol) [3] for service invocation. However, we enrich WSDL to allow an open specification of Quality of Service (QoS) parameters. Ariadne can operate on any IP network over 1-hop communication or multi-hop communication. To allow multi-hop communication Ariadne needs to run over an ad hoc routing protocol (e.g. OLSR, available at http://hipercom.inria.fr/olsr/).

II Ariadne functioning

Ariadne subdivides into two components: (i) Directories that are dynamically deployed on the network, and (ii) Service Discovery component (SD) that are deployed on all the nodes. The Directory caches the descriptions of networked services that have been discovered. The SD looks for directory instances in the vicinity of the local node and provides an interface that allows registering/unregistering services and searching services. When the SD receives register/unregister or search service requests, it forwards them to all the directories in its vicinity.

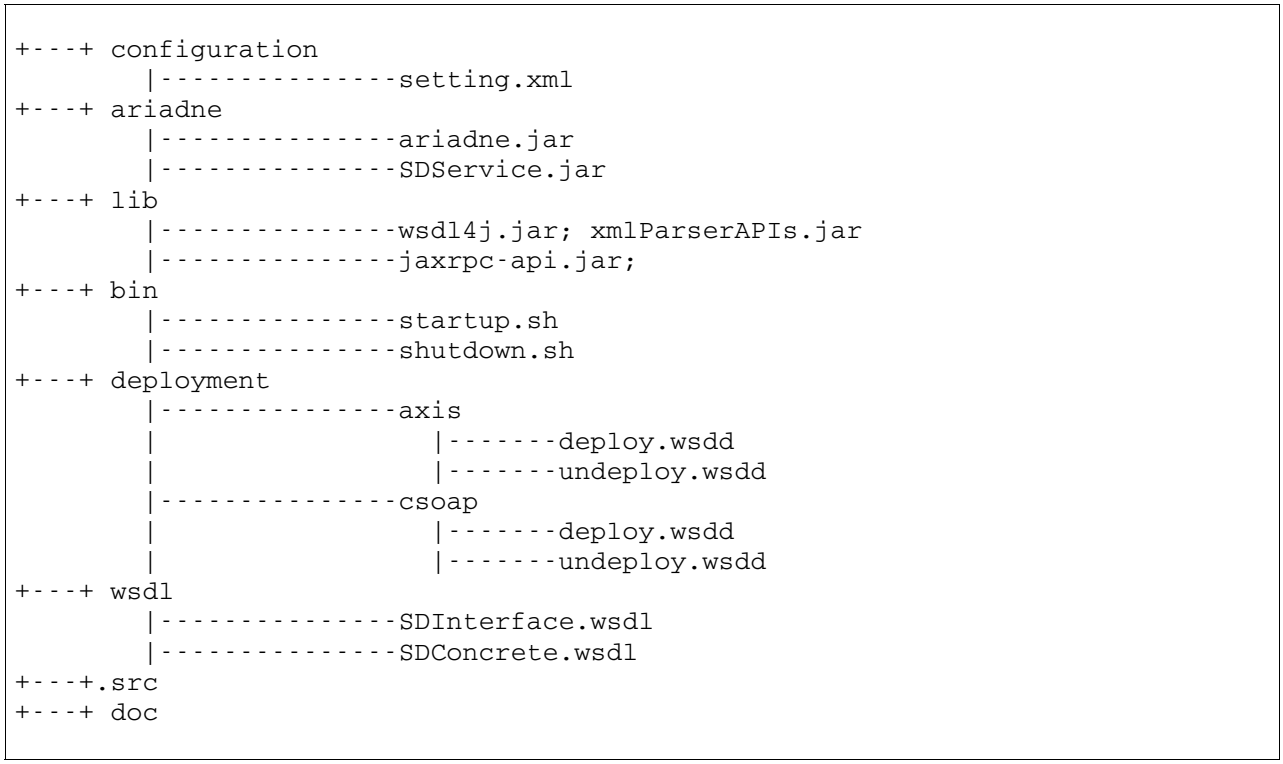
Each node in the network must run at least the SD component. The directory part can be deployed on some nodes in two ways: (i) by election, or, (ii) by forcing the deployment on the local node in the setting file (see Section IV.1).

The election process consists of choosing one or more nodes that can become directories. The election starts by broadcasting an election request from an SD without any directories in its vicinity, to all the other SD in its vicinity. An SD receiving the election request can either refuse or accept to act as a directory, depending on its capacity. When an SD is willing to act as a directory, it sends back an acceptance message, which contains its capacity (CPU, Memory, Disc, etc) and its connectivity (number of neighbors, number of neighbor directories, etc). Finally, the SD that initiated the election chooses one or more nodes to become directories based on received replies and sends them a notification message. The SD that receives a notification message deploys the directory component on the local node. The main selection criteria for directory hosts is the node coverage, i.e., the node that has the highest number of neighbor nodes and the smallest number of directories in its vicinity is chosen.

Each directory advertises its presence every T time unit. These advertisements are received by the SDs and the Directories in the vicinity. If an SD doesn't receive any directory advertisements after xT time (is a constant), it initiates the election process. All the directories exchange the summary of their cached services. These summaries are used by the directory to know which directory caches a WSDL description of a given service when it doesn't locally cache the service itself. In Ariadne, the directory summaries are generated with a bloom filter implementation from http://tiger.towson.edu/users/dknopp1.

III Installing Ariadne

The latest version of Ariadne can be found at http://www-rocq.inria.fr/arles/download/ariadne/. Download the ariadne-*{version}*.tar file and uncompress it into a temporary folder. You get the following file (Figure III):



1. Packages structure of Ariadne release

- configuration/setting.xml is an XML file used to configure Ariadne.
- ariadne/ariadne.jar contains the Ariadne release classes.
- ariadne/SDService.jar contains the SD component classes.
- lib/ contains the libraries needed by Ariadne.
- bin/ contains the scripts to start up and shut down Ariadne.
- deployment/axis/deploy.wsdd (undeploy.wsdd) is the deployment (undeployment) script of Ariadne on Apache Axis[1].
- deployment/csoap/deploy.wsdd (undeploy.wsdd) is the deployment (undeployment) script of Ariadne on CSoap[2].
- wsdl/SDInterface.wsdl is the WSDL description of Ariadne interface.
- wsdl/SDConcrete.wsdl is the concrete definition of Ariadne interface.
- src/ contains the Java source code of the Ariadne.
- doc/ contains this document.

Ariadne can be installed in two ways: (1) as a Web Service deployed on a SOAP container or (2) as a local part of the application that uses it. To install Ariadne as a Web Service, you have to install a Servlet container, we suggest you use Jakarta Tomcat (see http://jakarta.apache.org/tomcat/index.html for more detail); and a SOAP container, we suggest Axis or CSoap.

- *Installing Ariadne as a Web Service:*

The Ariadne release provides the deployment files of Ariadne as a Web Service on Axis (deployment/axis/deploy.wsdd), and on CSoap (deployment/csoap/deploy.wsdd). It also provides the Binding

[1] Apache Axis is an implementation of the SOAP. See http://ws.apache.org/axis/.

[2] CSoap is an implementation of the SOAP for resource-constrained devices. See http://www-rocq.inria.fr/arles/work/wsami.html.

files in the ariadne.jar, which are named SOAPBindingAxis.class for Axis and SOAPBindingCSoap.class for CSoap. If you want to use another SOAP container you have to write an appropriate deployment file and generate an appropriate Binding file from the provided SDInterface.wsdl and SDConcrete.wsdl files.

There are 4 steps to deploy Ariadne as a Web Service:
1/ copy the ariadne.jar and SDService.jar file into the shared folder of the Servlet container.
2/ copy the configuration folder into the default folder used by ariadne.jar under the Servlet container.
3/ copy the content of the lib folder into the shared folder of the Servlet container.
4/ use the wsdd deployment file appropriate to the SOAP container to deploy the Ariadne interface.

- *Installing Ariadne for local instance use:*
In this case, you only have to add the ariadne.jar and SDService.jar files and the content of the lib folder to the current *CLASSPATH* and copy the configuration folder into the default folder used by ariadne.jar.

IV Using Ariadne

Using Ariadne is the same for both types of installation. There is only a small difference in starting Ariadne, which will be explained in Section IV.2.
To be able to use Ariadne you have to know: the Ariadne setting file and the SD interface (its WSDL description is given in the annex). The setting file will be described in the next section. How to use the SD interface is explained in several sections according to the functionality provided (start, register service, unregister service, and find service, customization and QoS in the QoS management Section).

IV.1 Setting file

The Ariadne setting file is named *setting.xml.* You can find it in the configuration folder and you must put it into the current folder used by Ariadne. The setting file contains a set of XML targets and their values, which are responsible for the proper functioning of Ariadne. To describe the XML targets we will show and explain the default configuration of the setting file (see Figure IV.1). Only experienced users should modify this default configuration according to their network topology and connectivity.

```
<?xml version="1.0" ?>
  <ServiceDiscovery>
    - <!--   Directory deployment values auto, false, true -->
    -<!--  auto  : With election process if any -->
    - <!--  false : never elected as directory -->
    - <!--  true  : always elected as directory -->
    <DirectoryDeployment>auto</DirectoryDeployment>
    <MinPerformance>0.5</MinPerformance>
    <HopsNumber>1</HopsNumber>
    <MaxDirectoryRequested>5</MaxDirectoryRequested>
    <GetServiceRequestHops>2</GetServiceRequestHops>
    <PortNumber>9999</PortNumber>
    <BloomFilter>
       - <!--  activation values : true , false -->
      <activation>true</activation>
      <size>2800</size>
      <hashFunctionNumber>4</hashFunctionNumber>
    </BloomFilter>
    <Cache>
      - <!--  activation values : true , false -->
      <activation>true</activation>
      <size>100</size>
    </Cache>
    - <!--  ProfilePort value must be different of PortNumber value defined above -->
    <ProfilePort>5555</ProfilePort>
    <QoS>
      - <!--  activation values : true , false -->
      <activation>true</activation>
    </QoS>
    <TimingValues>
    <SD>
      <LISTENING_PERIOD_VALUE>5</LISTENING_PERIOD_VALUE>
      <ADVERTISEMENT_PERIOD_VALUE>5000</ADVERTISEMENT_PERIOD_VALUE>
      <ELECTION_WAITING_RESPONSE_TIME_VALUE>6000</ELECTION_WAITING_RESPONSE_TIME_VALUE>
      <WAITING_GET_SERVICE_RESPONSE_MIN_VALUE>50</WAITING_GET_SERVICE_RESPONSE_MIN_VALUE>
      <WAITING_GET_SERVICE_RESPONSE_MAX_VALUE>20000</WAITING_GET_SERVICE_RESPONSE_MAX_VALUE>
      <CHECKING_PERFORMANCE_PERIOD>60000</CHECKING_PERFORMANCE_PERIOD>
    </SD>
    <Election>
      <ELECTION_HOPS_VALUE>1</ELECTION_HOPS_VALUE>
      <LISTENING_PERIOD_VALUE>5</LISTENING_PERIOD_VALUE>
      <ADVERTISEMENT_PERIOD_VALUE>5000</ADVERTISEMENT_PERIOD_VALUE>
      <ELECTION_WAITING_RESPONSE_TIME_VALUE>6000</ELECTION_WAITING_RESPONSE_TIME_VALUE>
    </Election>
    <Directory>
      <LISTENING_PERIOD_VALUE>5</LISTENING_PERIOD_VALUE>
      <ADVERTISEMENT_PERIOD_VALUE>5000</ADVERTISEMENT_PERIOD_VALUE>
      <ADVERTISEMENT_HOPS_VALUE>2</ADVERTISEMENT_HOPS_VALUE>
      <SUMMARIE_EXCHANGE_PERIOD_VALUE>20000</SUMMARIE_EXCHANGE_PERIOD_VALUE>
      <WAITING_GET_SERVICE_RESPONSE_MIN_VALUE>50</WAITING_GET_SERVICE_RESPONSE_MIN_VALUE>
      <WAITING_GET_SERVICE_RESPONSE_MAX_VALUE>10000</WAITING_GET_SERVICE_RESPONSE_MAX_VALUE>
      <INITIAL_TTL_VALUE>30000</INITIAL_TTL_VALUE>
      <TTL_UPDATING_VALUE>3000</TTL_UPDATING_VALUE>
    </Directory>
    </TimingValues>
  </ServiceDiscovery>
```

- *DirectoryDeployment*: takes three values: auto, false and true.
  auto: Ariadne deploys a directory on the local node if and only if an elect message is received.
  false: Ariadne will never deploy a directory in the local node and reply with a refuse message for all election requests received.
  true: Ariadne deploys a directory on the local node even when it starts up, independently of the directories deployed in its vicinity.

- *MinPerformance*: takes a numeric value between 0 and 1. This value is the percentage of the node's capacity. If the current capacity of the local node is less than the *MinPerformance* value the `Ariadne` can replies with a refuse message to the election requests, if not, it can send its candidature.

- *HopsNumber:* is the number of hops over which the node searches for its neighbor nodes. This value must be higher than, or equal to 1, and should be as small as possible, in order to avoid flooding the network.

- *MaxDirectoryRequested:* is the maximum number of the directories that the SD may request when it searches for a service.

- *GetServiceRequestHops:* is the maximum number of hops that a search message can make between the directories.

- *PortNumber:* defines the port number that is used to send and receive all Ariadne messages except the control messages which are used to calculate the number of the neighbor nodes.

- *BloomFilter:* contains three sub-targets: *activation*, *size* and hashFunctionNumber
  - *Activation*: takes two values: true or false. The true value activates the generation of the service summary and their exchange with the other directories, and the false value deactivates it.
  - *Size*: is the size of the array that contains the services' summaries. See [1] for more details.
  - *HashFunctionNumber:* is the number of the values given by the hash function. See [1] for more details.

- *Cache:* contains two sub-targets: *activation* and *size that* concern the cache. The cache is an array where the service descriptions found in the remote directories are temporarily stored. The cache may be used to answer other search requests.
  - *Activation:* takes two values: true or false. The true value activates the cache use and the false value deactivates it. To increase the Ariadne performances we suggest you activate the cache.
  - *Size:* is the size of the cache array.

- *ProfilePort:* defines the port number that is used to send and receive the control messages, which are used to calculate the number of the neighbor nodes. The *ProfilePort* number must be different from the *PortNumber* value defined above.

- *QoS:* contains one target: *activation*
  - *Activation*: takes two values: true or false, that specify respectively whether Ariadne takes care of the quality of service or not. See section IV.6.1 for more details.

TimingValues

This target contains the list of the waiting times and the frequency advertisement times of the SD, the Directory, and the election process. The default values of the waiting times are defined according to a MANET formed by five laptops, which have CPU: Intel PII 800MHz, RAM: 192Mo and wireless card: Lucent WaveLAN PC Card 11Mb/s.

For SD

- *LISTENING_PERIOD_VALUE*: is the period of time for the SD to take the received messages from the message stack.

- *ADVERTISEMENT_PERIOD_VALUE*: is the frequency time of the directory advertisement.

- *ELECTION_WAITING_RESPONSE_TIME_VALUE*: is the time that the SD must wait when the election is processing. This time will be multiplied by the number of hops.

- *WAITING_GET_SERVICE_RESPONSE_MIN_VALUE*: when the SD sends a search service message, it looks for the reply message every *WAITING_GET_SERVICE_RESPONSE_MIN_VALUE* time.

- *WAITING_GET_SERVICE_RESPONSE_MAX_VALUE*: if the SD doesn't receive a response message for a search service message after the *WAITING_GET_SERVICE_RESPONSE_MAX_VALUE*, it returns an empty result.

- *CHECKING_PERFORMANCE_PERIOD:* is the frequency time for checking performance value of the local node.

For Election

- *ELECTION_HOPS_VALUE:* is the number of hops over which an election message is sent.
- *LISTENING_PERIOD_VALUE:* is the period of time for the election process to take the received messages from the message stack.

- *ADVERTISEMENT_PERIOD_VALUE:* is the frequency time of the directory advertisement.

- *ELECTION_WAITING_RESPONSE_TIME_VALUE:* is the time that the Election process must wait to receive the election response messages. This time will be multiplied by the number of hops.

For directory

- *LISTENING_PERIOD_VALUE:* is the period of time for the Directory to take the received messages from the messages stack.

- *ADVERTISEMENT_PERIOD_VALUE:* is the period of time for the directory to advertise its presence.

- *ADVERTISEMENT_HOPS_VALUE*: is the number of hops over which the directory sends its advertisements
- *SUMMARIE_EXCHANGE_PERIOD_VALUE*: is the period of time for the directory to send its services' summary to the other directories, if its summary has been changed.

- *WAITING_GET_SERVICE_RESPONSE_MIN_VALUE*: when the Directory sends a search service message to the other directories, it looks for the reply message every *WAITING_GET_SERVICE_RESPONSE_MIN_VALUE* time.

- *WAITING_GET_SERVICE_RESPONSE_MAX_VALUE*: if the Directory doesn't receive a response message for a search service message after the *WAITING_GET_SERVICE_RESPONSE_MAX_VALUE*, it returns an empty result.

- *INITIAL_TTL_VALUE*: is the initial living time for a received directory advertisement.
- *TTL_UPDATING_VALUE*: is the period of time for the directory to update the TTL of other directory advertisements. Then the directory removes them when their TTL is equal to zero.

IV.2 Starting Ariadne

If Ariadne is deployed as a Web Service you can use the bin/startup.sh provided in the release to start Ariadne and bin/shutdown.sh to stop it. In your application you can obtain an instance of the Ariadne by using the Binding classes as shown in the following example (Figure IV.2).

```
java.net.URL localSDURL =
       new java.net.URL("http://localhost:8080/service/SD");
    SDService.SDPortType sd = new SDService.SDBindingStub(localSDURL,null);
```

3. Getting an instance of Ariadne

In the other case, to start Ariadne, you have to create an instance of the servicediscovery.SD.class in your application and you invoke the start() method to start Ariadne. The isStarted() method of the servicediscovery.SD.class returns true when Ariadne is ready to use and returns false otherwise. The following is an example of the Ariadne starting with instantiation (Figure IV.2).

```
Import SDService.SDImpl;
…….
Public class myApplication
{
….

SDService.SDPortType sd = new SDImpl();
Sd.start();
While (!sd.isStarted()) {}
// Ariadne is ready to use
….
}
```

4. Example of the Ariadne starting with instantiation

IV.3 Registering a service

As described above, Ariadne main role is to allow service providers to advertise their services and to allow clients to find them. The service advertisement is enabled by the register function given by the SD. The register function has four parameters: the WSDL description of the service, the endpoint of the service URL, the WSDL description of the local customizer, the WSDL description of the remote customizer. In this section we will show only the use of the first and second parameters, the customizers parameters will be described in Section IV.6.2.
The following (Figure IV.3) is an example of the register function:

```
// SD starting : see the starting section

// Part 1
// get the wsdl description from the wsdl file
String wsdlFilePath = "/servicesproviders/services/wsdl/MyServiceInterface.wsdl" ;
String wsdlDescription = "";

Try
   {
     java.io.FileInputStream wsdlFile = new java.io.FileInputStream(wsdlFilePath);
      while (wsdlFile.available() >0)
                {
                   byte[] data = new byte[wsdlFile.available()];
                   wsdlFile.read(data);
                   wsdlDescription = wsdlDescription+(new String(data));
                }
    }
catch(FileNotFoundException ex)
{
System.out.println(wsdlFilePath+" file not found") ;
}
catch(IOException ex)
{
System.out.println(wsdlFilePath+" file I/O Error") ;
}

// Part 2
// create the EndPoint URL

String hostAddress = "128.10.10.10" ; // the address of the host where the service is
deployed
String port = "8080" ;
String servicesContext = "services"
String serviceName = "myService" ;

String                                    EndPointURL                                    =
"http://"+hostAddress+" :"+port+"/"+servicesContext+"/"+serviceName;
// EndPointURL = http://128.10.10.10:8080/services/myService;

// Part 3
// now you can register the  service
sd.register (wsdlDescription, EndPointURL, "", "");
```

5. The register function

There are 3 parts in the above example:

Part1: we obtain the WSDL description from the file, and put it into a string.

Part2: we create the EndPoint URL from the set of information (host address, port number, service context and service name). The host address is the address of the host where the service is deployed. In this case, the host has the address 128.10.10.10 and the service's name is *myService* which is deployed in the *services* context.

Part3: we call the register function and give it the WSDL description of the service and its EndPoint URL.

IV.4 Unregistering a Service

A service is unregistered by unregister function given by the SD. This function has two parameters : the WSDL description of the service to remove and its EndPoint URL. When the unregister function is invoked the service is removed from all the directories. Figure IV.4 shows an example of the unregistering of the service that was registered in the previous section (Figure IV.3).

```
// get the wsdl description from the wsdl file
String wsdlFilePath = "/servicesproviders/services/wsdl/MyServiceInterface.wsdl" ;
String wsdlDescription = "";

Try
   {
     java.io.FileInputStream wsdlFile = new java.io.FileInputStream(wsdlFilePath);
       while (wsdlFile.available() >0)
                 {
                    byte[] data = new byte[wsdlFile.available()];
                    wsdlFile.read(data);
                    wsdlDescription = wsdlDescription+(new String(data));
                 }
    }
catch(FileNotFoundException ex)
{
System.out.println(wsdlFilePath+" file not found") ;
}
catch(IOException ex)
{
System.out.println(wsdlFilePath+" file I/O Error") ;
}

EndPointURL = "http://128.10.10.10:8080/services/myService";

// call unregister function
sd.unregister (wsdlDescription, EndPointURL);
```

6. The unregister function

IV.5 Finding a service

Ariadne aims to provide clients with an efficient means to find services, by optimizing the response time, the network flooding and finding the services that match the clients' specifications. To find a service you have to give Ariadne a full WSDL description or a partial WSDL description of the required service. Ariadne searches for the given service description in the directory that is most likely caches the service. Finding the service is enabled by the getServices() function of the SD, which has one parameter (WSDL description) and returns an array of GetServiceData objects. Each GetServiceData object contains a description of a service that has been found. The structure of GetServiceData class is shown in Figure IV.5.

```
Package SDService;

Public class GetServiceData  implements java.io.Serializable {
    Private java.lang.String serviceEP;
    Private java.lang.String custLocalEP;
    Private java.lang.String custRemoteEP;
    Private java.lang.String wsdlDocument;
    Private java.lang.String wsdlCustLocal;
    Private java.lang.String wsdlCustRemote;

    Public GetServiceData() {
    }

  // setter and getter methods of all the attributes
```

7. GetServiceData Class

serviceEP: is the service's EndPoint.
custLocalEP: is the EndPoint of the local customizer, if any.
custRemoteEP: is the EndPoint of the remote customizer, if any.
wsdlDocument: is the full WSDL description of the service.
wsdlCustLocal: is the full WSDL description of the local customizer service, if any.
wsdlCustRemote: is the full WSDL description of the remote customizer service, if any.

Generally, the service search in Ariadne is composed of four parts:
1) Getting the full/or partial WSDL description of the required service.
2) Asking the Ariadne to search for the registered services whose WSDL description contains a given WSDL description.
3) Selecting one or more of the found services.
4) Using the found service EndPoints to connect to the services and use them.

These four parts are shown in the following example (Figure IV.5).

```
// Part 1
// get WSDL description of the required service
String partialWSDLDescription = "";
String partialWSDLFilePath = "/clients/wsdl/wsami/MyServiceInterface.wsdl";

Try
{
    Java.io.FileInputStream wsdlFile = new
                                        java.io.FileInputStream(partialWSDLFilePath);

    While (wsdlFile.available() >0)
       {
         byte[] data = new byte[wsdlFile.available()];
         wsdlFile.read(data);
         partialWSDLDescription = partialWSDLDescription +(new String(data));
       }
}
catch(...)
{
...
}
// Part 2
// find the services
SDService.GetServiceData[] services = sd.getServices(partialWSDLDescription);

// Part 3
// treat the found services
if (services != null && services.length > 0)
  {
    //take the first service found
    String serviceEndPoint = services[0].getServiceEP();
    String localCustomizerEndPoint = services[0].getCustLocalEP();
    String remoteCustomizerEndPoint = services[0].getCustRemoteEP();
  }

// Part 4
// use the selected service
java.net.URL serviceURL =  new java.net.URL(serviceEndPoint);
ServiceInterface service = new Service.ServiceBindingStub(serviceURL,null);
```

8. Finding a service

IV.6 QoS Management in Ariadne

Ariadne provides two optional mechanisms to handle the QoS-awareness. The first is the QoS Parameters, which allow the user to find the appropriate service, and the second is the Customization, which allows the user to customize its service running. The following describes these two mechanisms.

IV.6.1 QoS Parameters

We enrich the WSDL language by integrating QoS parameters, which are mainly used to allow the clients to find the appropriate service according to the context. The service's QoS parameters and their values are initially given by the service provider when it registers the service, and the values are updated periodically by Ariadne. The client requiring a service can add some QoS parameters and their values into the WSDL description of the required service, these values will be compared to the values given by the service provider. Only the services, that offer at least the QoS values equal to or higher than the QoS values requested by the client, are selected.

The following (Figure IV.6.1) is the XML Schema of the QoS parameters.

```
<xs:element name="ServiceDiscovery">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ttl" type="xs:integer" default="-1"/>
      <xs:element name="QoS">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="param" minOccurs="0" maxOccurs="10">
              <xs:complexType>
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="type" type="xs:integer"/>
                <xs:attribute name="ttl" type="xs:integer"/>
                <xs:attribute name="value" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

9. QoS parameters XML Schema.

The root element is the ServiceDiscovery, and it contains two elements: ttl and QoS. The ttl element is the time to live of the service description registered in the directory. Once this time has expired the service description is removed from the directory. To keep a service description in the directory all the time, the service provider has to re-register the service before its ttl expires or give the –1 to the ttl value. The QoS element contains an occurrence of the param element, which allows the user to define any QoS attribute and give it a value. The param element has four attributes: name, type, ttl and value.

Name: is the name of the QoS parameter.

Type: is the type of the QoS parameter, 0 for integer type, and 1 for string type. We will add other types in future versions.

ttl: is the time to live of the QoS parameter. When the ttl expires the QoS value is set to null. To keep the QoS parameter continually put its ttl value to –1.

Value: is the value of the QoS parameter, and may be a string or an integer.

The Figure IV.6.1 shows an example of QoS parameters

```
<ServiceDiscovery>
<ttl>120000</ttl>
<QoS>
<param name = instance     type = 1 ttl = -1     value = www-rocq.inria.fr/arles/>
<param name = energy       type = 0 ttl = 60000  value = 45/>
<param name = memory       type = 0 ttl = 60000  value = 64000/>
<param name = freememory   type = 0 ttl = 60000  value = 5000/>
<param name = cpu          type = 0 ttl = 60000  value = 800/>
<param name = cachesize    type = 0 ttl = 60000  value = 256/>
<param name = connectivity type = 0 ttl = 80000  value = 5/>
<param name = bandwith     type = 0 ttl = 90000  value = 50/>
</QoS>
</ServiceDiscovery>
```

10. Example of QoS parameters

The comparison between the QoS parameters required by a client (Client QoS) and the QoS parameters given by a service provider (Provider QoS) return true if and only if:

- each parameter of the client QoS is contained in the provider QoS (name based).
- all the ttls of the Client QoS are equal to or higher than their corresponding ttls of the Provider QoS.
- all the values of the Client QoS are equal to their corresponding values of Provider QoS, in the case of the string type.
- all the values of the Client QoS are equal to or higher than their corresponding values of the Provider QoS, in the case of the integer type.

IV.6.2 Customization

Our approach in service Customization is inspired by WSAMI Customization [4], which is based on using of a pair of content customizers at both ends, as presented in [5]. A Customizer consists of a local and a remote service. Any message exchanged between the client and the service goes through the Customizer. See http://www-rocq.inria.fr/arles/work/wsami.html for more details.

As shown above, in register service (section IV.3), Ariadne registers the WSDL description of a service and can also register a WSDL description of the local service customizer and a WSDL description of the remote service customizer related to the service. These customizers can be used to customize the running of the service. When a client asks the Ariadne for a registered service, the Ariadne searches and gives the EndPoints of the service and also the EndPoints of its customizers. Figure IV.6.2 shows an example.

```
SDService.GetServiceData[] services = sd.getServices(partialWSDLDescription);

if (services != null && services.length > 0)
  {
    //take the first service found
    String serviceEndPoint = services[0].getServiceEP();
    String localCustomizerEndPoint = services[0].getCustLocalEP();
    String remoteCustomizerEndPoint = services[0].getCustRemoteEP();
  }
```

11. Example of finding service customizers

V Comparison between service discovery protocols

The following table (Figure V) shows a comparison between some well-known service discovery protocols and our Ariadne implementation, listing which functionalities are supported in each case. The table comes from [6] except the Ariadne column.

	Bluetooth	**DEAPspace**	**INS**	**Jini**	**Salutation**	**SLP**	**SSDS**	**UPnP**	**Ariadne**
Service									
Naming and attributes	Standard	N/A	N/A	Standard	Standard	Standard	N/A	Standard	WSDL
Invocation	N/A	N/A	N/A	Java code	Remote Procedure Call	URL	N/A	XML Data	SOAP
Status inquiry	N/A	N/A	N/A	Notification and event agent	Notification	N/A	N/A	Polling and notification	N/A
Directory									
Centralized vs distributed	N/A	N/A	Distributed	Distributed	Either	Centralized	Distributed	N/A	Distributed
Number of Service Information Copies	N/A	N/A	Fully replicated in sub domains, single copy globally	Multiple copies	Multiple copies	Multiple copies	Single copy	N/A	Multiple copies
Flat vs. Hierarchical	N/A	N/A	Flat and hierarchical	Flat or hierarchical	Flat	N/A	Hierarchical	N/A	Flat
Service State in directories	N/A	N/A	Soft state and hard state	Soft state	Hard state with periodically check	Soft state	Soft state and hard state	N/A	Soft state
Directory address	N/A	N/A	Configured address	Configured or multicast address	Configured or multicast address	Multicast address	Multicast address	N/A	Discovered address
Number of directory hierarchies	N/A	N/A	Multiple hierarchies	Single hierarchy	N/A	N/A	Multiple hierarchies	N/A	N/A
Announcement and lookup									
Query vs. Announcement	Query	Announcement	Both	Both	Both	Both	Both	Both	Query
Directory-based vs. non-directory-based	Non-directory-based	Non-directory-based	Directory-based	Directory-based	Directory-based	Either	Directory-based	non-directory-based	Directory-based
Communication	Unicast and broadcast	Broadcast	Unicast, anycast and multicast	Unicast and multicast	Unicast and broadcast	Multicast	Unicast, multicast and broadcast	Unicast and multicast	Unicast
Service Selection									
User vs. Protocol Selection	User selection	User selection	Protocol selection	User selection	User selection	User selection	User selection	User selection	User selection
Service Matching	Match all	N/A	Match best	Match all	Match one or match all	Match all	Match all or match best	Match all	N/A
Context-aware	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Scope-aware	Location / vicinity	N/A	Location and administrative domain	Location and administrative domain	N/A	Administrative domain	Location, administrative domain and network topology	N/A	N/A

QoS-aware	N/A	N/A	Yes	N/A	N/A	N/A	N/A	N/A	Yes

12. Comparison between service discovery protocols

VI Appendix

WSDL Description of SD Interface

```
<?xml version="1.0" ?>
<definitions name="urn:SDService"
             targetNamespace="urn:SDService"
             xmlns:tns="urn:SDService"
             xmlns:typens="urn:SDService"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- type defs -->
  <types>
    <xsd:schema targetNamespace="urn:SDService">

     <xsd:complexType name="GetServiceData">
       <xsd:sequence>
        <xsd:element name="serviceEP" type="xsd:string"/>
        <xsd:element name="custLocalEP" type="xsd:string"/>
        <xsd:element name="custRemoteEP" type="xsd:string"/>
        <xsd:element name="wsdlDocument" type="xsd:string"/>
        <xsd:element name="wsdlCustLocal" type="xsd:string"/>
        <xsd:element name="wsdlCustRemote" type="xsd:string"/>
       </xsd:sequence>
     </xsd:complexType>

     <xsd:complexType name="ArrayOfGetServiceData">
       <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
                  <xsd:sequence>
                   <xsd:element name="strings" minOccurs="0"
                    maxOccurs="unbounded"
                     type="typens:GetServiceData"/>
          </xsd:sequence>
       </xsd:restriction>
       </xsd:complexContent>
     </xsd:complexType>
    </xsd:schema>
  </types>

  <!-- message declns -->
  <message name="startRequest"/>
  <message name="startResponse">
     <part name="stat" type="xsd:boolean"/>
  </message>
  <message name="stopRequest"/>

  <message name="RegisterRequest">
```

```
   <part name="WSDL" type="xsd:string"/>
   <part name="EP" type="xsd:string"/>
   <part name="WSDLLCU" type="xsd:string"/>
   <part name="WSDLRCU" type="xsd:string"/>
</message>
<message name="UnRegisterRequest">
  <part name="WSDL" type="xsd:string"/>
  <part name="EP" type="xsd:string"/>
</message>

<message name="GetServices_ByInstanceURLRequest">
  <part name="WSDL" type="xsd:string"/>
  <part name="instanceURL" type="xsd:string"/>
</message>

<message name="GetServices_ByInstanceURLResponse">
  <part name="serviceEP" type="typens:ArrayOfGetServiceData"/>
</message>

<message name="GetServicesRequest">
  <part name="WSDL" type="xsd:string"/>
</message>

<message name="GetServicesResponse">
  <part name="serviceEP" type="typens:ArrayOfGetServiceData"/>
</message>

<message name="GetAllServicesRequest">
  <part name="GroupServiceWAU" type="xsd:string"/>
</message>

<message name="GetAllServicesResponse">
  <part name="serviceEP" type="typens:ArrayOfGetServiceData"/>
</message>

<!-- port type declns -->
<portType name="SDPortType">

  <operation name="start">
    <input message="tns:startRequest"/>
    <!-- output message="tns:startResponse"/ -->
  </operation>

  <operation name="stop">
    <input message="tns:stopRequest"/>
  </operation>

  <operation name="Register">
    <input message="tns:RegisterRequest"/>
  </operation>

  <operation name="UnRegister">
    <input message="tns:UnRegisterRequest"/>
  </operation>

  <operation name="getServices">
    <input message="tns:GetServicesRequest"/>
```

```
  <output message="tns:GetServicesResponse"/>
</operation>
```

```
  <operation name="getAllServices">
    <input message="tns:GetAllServicesRequest"/>
    <output message="tns:GetAllServicesResponse"/>
  </operation>

  <operation name="getServices_ByInstanceURL">
    <input message="tns:GetServices_ByInstanceURLRequest"/>
    <output message="tns:GetServices_ByInstanceURLResponse"/>
  </operation>

 </portType>


<!-- binding declns -->
<binding name="SDBinding" type="tns:SDPortType">

  <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http" />

  <operation name="start">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:SDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>


  <operation name="stop">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:SDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>


  <operation name="Register">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:SDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>
  <operation name="UnRegister">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
                 namespace="urn:SDService"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>

  </operation>
  <operation name="getServices">
```

```
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>


   <operation name="getAllServices">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="getServices_ByInstanceURL">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:SDService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

  </binding>

</definitions>
```

VII References

[1] F. Sailhan, V. Issarny. Scalable Service Discovery for MANET. In proc of IEEE International Conference on Pervasive Computing and Communications (PERCOM), March 2005, accepted paper.

[2] Web Services Description Working Group. http://www.w3c.org/2002/ws/desc/.

[3] XML Protocol Working Group. http://www.w3c.org/2000/xp/Group/.

[4] D. Sacchetti, R. Chibout, V. Issarny. WSAMI: A Middleware Infrastructure for Ambient Intelligence based on Web Services. http://www-rocq.inria.fr/arles/work/wsami.html.

[5] J. Steinberg and J. Pasquale. A Web middleware architecture for dynamic customization of content for wireless clients. In Proceedings of the WWW'02 Conference, 2002.

[6] F. Zhu, M. Mutka, and L. Ni. Classification of Service Discovery in Pervasive Computing Environments. *MSU-CSE-02-24*, Michigan State University, EastLansing, 2002.