Web Service Integration Gateway

Table of contents

1 Installation.....2
1.1 Web Service Integration Gateway's Installation.....2
2 Tutorial.....5
2.1 Web Service Integration Gateway, Programmer's Tutorial.....5
3 PDFs.....21

1. Installation

1.1. Web Service Integration Gateway's Installation

1.1.1. Introduction

This document describes steps how is the project installed. The external third party software installation is described in the first sections. The WSIG's installation is described in the section Installation.

1.1.2. Products Required

The project requires the following products to be installed:

- JAVA 1.4,
- Apache ANT,
- Apache FORREST,
- JADE 3.3,
- AXIS,
- an XML's parser for AXIS,
- WSDL's library,
- log4j,,
- uddi4j,,
- access into a UDDI registry.
-

The list item is bound with corresponding section, where an installation is described. Information is included about reasons in sections, what is the purpose of a specific product.

1.1.2.1. JAVA's Installation

The Java installation notes are available at JAVA-1.4 home page. The installation steps are described in links per an architecture on the page.

The required version of the JAVA is 1.4 for the WSIG. A new JAVA version 1.5 has a problem with AXIS 1.1.

1.1.2.2. Apache Ant's Installation

The Apache Ant is needed to build the WSIG project. The Ant's documentation describes installation's steps.

1.1.2.3. Apache Forrest's Installation

The documentation is managed by the Apache Forrest. The project Apache Forrest may

be downloaded from Forrest home page. An installation are provided with the package.

1.1.2.4. JADE's Installation

The JADE's package is located at JADE's site. Installation's details are packaged in the JADE. The WSIG is add on software for the JADE. The JADE 3.3 is tested for this release of the WSIG.

1.1.2.5. AXIS's Installation

SOAP's messages are handled by the soap manipulation interface. The AXIS project provides the implementation of the interface. The AXIS-1.1 is used currently. After downloading, .jar files are placed into lib/axis-1.1 directory.

The version 1.1 requieres the JAVA in version 1.4. A new AXIS version may be runnable in JAVA >1.4.

An XML's parser is needed for the AXIS. The Xerces-2.6.2 is one of the free available. Xerces .jar's files are placed into lib/xerces-2_6_2 directory.

1.1.2.6. WSDL Library's Installation

The WSDL's files describe a Web Service. The WSDL4j-1.4 is used to manipulate with this format from JAVA programms. After downlading, the .jar file is placed into lib/wsdl4j-1.4 directory.

1.1.2.7. Log4j's Installation

The logging is done by log4j project. Actually, the .jar file is provided by the AXIS and is located in directory lib/axis-1_1/lib.

1.1.2.8. The uddi4j's Installation

An access into a UDDI's registry is done by uddi4j project. The project provides a JAVA access into a UDDI registry. After downlading, the .jar file is placed into lib/uddi4j-2_0_2 directory.

Note: up to now (2005 Nov) the uddi4j is not accessible. It seems a new acces method into UDDI is established. See technical not Generating a JAX-RPC Client for UDDI 3.0.2 for details. A local copy of uddi4j-bin-2_0_2.zip.

1.1.2.9. The UDDI Registry

An access into a UDDI v2.0 registry is required to provide information about JADE's services registered into the WSIG. The external Web Service client and server does requests for an access information. The Web Service side contacts the UDDI at first.

The jUDDI implementation with the mysql database may be used as a UDDI registry. These products are available free. The juddi-0.9rc3 is tested. The installation instructions are provided with jUDDI package. They are more complex and there is small text document jUDDI_install.txt related to WSIG's requirements only. (Please, create your publisher information and create your business records. Odervice, a connection will be rejected.)

1.1.3. Installation

The product is extracted into JADE directory add-on. Relevant products have been installed as is mentioned in section Products Required. The package comes with a .jar library precompiled. A new recompilation may be initiate by running `ant jar` in WSIG's directory. A documentation is able to be recompiled by typing `forrest` in doc directory. A result is stored into doc/build directory.

1.1.3.1. WSIG's Configuration

The configuration file wsig.properties is lacated in directory, where the WSIG runs typically. The original one is placed in misc directory. The correct values are edited in this file. It is suitable to change lines with properties followed:

wsig.host.name
the host name of the machine, where the WSIG runs, (`wsig.host.name=pc7`)
wsig.host.port
a port number, where a gateway will be worked at (`wsig.host.port=2222`)
wsig.agent_id
agent ID of a gateway, the host name part must be changed according machine's host name. To leave a host name as localhost brings identification problems. (`wsig.agent_id=wsigs@pc7\:1099/JADE`)
uddi.wsig_businessKey
a business key of your business entity in a UDDI repository (`uddi.wsig_businessKey=8C983E50-E09B-11D8-BE50-DA8FBF3BDC61`)
uddi.userName, uddi.userPassword
a user name and a password into the UDDI repository (`uddi.userName=jna` and `uddi.userPassword=somepasword`)
uddi.lifeCycleManagerURL
access point into the UDDI repository (`uddi.lifeCycleManagerURL=http\://localhost\:8080/jUDDI/publish`)
uddi.queryManagerURL
access point into the UDDI repository (

```
uddi.queryManagerURL=http\://localhost\:8080/jUDDI/inquiry
)
```

Startup files require a change, when they are running outside a localhost. Please, change host name in wsig.bat, run_TestAgentClient.bat, and run_TestAgentServer.bat. The script's name ends with the extension .sh under the UNIX.

A level of debug messages are set in log4j.properties file. The misc/log4j.properties.txt may be copied into running directory. The extension .txt must be cut off. It may be changed then.

1.1.3.2. WSIG's Running

Scripts must be run in the following order:

1. `run_jade_main.bat`
2. `wsig.bat`

or under the UNIX:

1. `run_jade_main.sh`
2. `wsig.sh`

It is possible to start WSIG's GUI. Please, edit wsig.bat or wsig.sh script to append "-gui" parameter after last command in the script. There are some examples available by menu selection. They are provided as tuples: a registration and an invocation. At first, the registration one must be run and then invocation is possible to perform. The google search example requires to enter your google's ID into the wsig.properties.

1.1.3.3. WSIG's Examples

Examples for testing purpose are placed in misc directory. Unix versions have extension .sh insted of .bat. After running TestSOAPServer the TestAgentClient invokes a service.

1. run_TestSOAPServer.bat
2. run_TestAgentClient.bat

The second examples demonstrate an agent service, which is invoked by Web Service client.

1. run_TestAgentServer.bat
2. run_TestSOAPClient.bat

2. Tutorial

2.1. Web Service Integration Gateway, Programmer's Tutorial

2.1.1. Introduction

The tutorial describes an usage of the Web Service Integration Gateway (WSIG). There

are an assumption, a reader has knowledge of programming the JADE agents and Web Services as stand alone.

Two directions are provides by the WSIG. JADE agent's services may be registered and provides by the WSIG as Web Services(WS). Servers for Web Services may register themselves into the WSIG and be accessible to JADE's agents.

2.1.2. The Functional Description

The WSIG's installation is considered to be finished. Servers for both sides is required to be registered into the WSIG then. Registration's steps are specific on each side. Clients search and invoke services of the servers then.

2.1.2.1. The JADE Agent Services' Exposition

A JADE's agent has intention to provide some of it's services for Web Services' clients. It must register itself into Directory Facilitator (DF) on JADE platform running. The DF sends all registration's requests into the WSIG. To identify services, which will be exposed for WS clients by the WSIG, the type of services must be set as "web-service". The "type" slot for service's description may be used as a main place for such information. A property "type" of the service may be used in case, when the agent already have some type. The property is set of possible types and "web-service" is a member. Services labelled by "web-service" are exposed to WS side. The WSIG creates a new unique operation name for the service. A WSDL structure is created as data type for a SOAP. A UDDI repository is requested for registration of the new operations provided by the WSIG.

A WS clients search through the UDDI repository for an operation required. An access point is stored in a UDDI's record returned. It is the access point provided by the WSIG. A client constructs a SOAP request and sends one to the WSIG's access point. The WSIG translates the request into an ACL SL0 message with name of an operation translated into a service's name of an agent. The ACL's message are sent into an agent. A response of the agent is translated back into a SOAP format and is sent to the client.

2.1.2.2. The WS Operations' Exposition

A WS server has operations, which are wanted to expose as JADE services. It must register it's operations into the WSIG. The WSIG provides a UDDI interface like a full functional UDDI repository. All UDDI's requests are forwarded into a UDDI repository configured. Any registration related requests are identified and processed by the WSIG. A WS server must send a WSDL information in a registration. The WSDL is parsed by the WSIG and all operations are identified. New JADE services' names are generated for these operations. The WSIG registers services as it's own ones into a JADE DF. Relevant information from a UDDI records are stored in service's properties. (an access point, an original name)

A JADE agent-client searches a DF for a service. It may find a service provided by the WSIG. It constructs an ACL SL0 message, fills a service name, and sends the message to the WSIG. The WSIG receives the message. A service name is translated into an operation and an access point. The message is translated into a SOAP message with the name of the operation. Such message is sent into the access point of a WS server. An answer from the server is translated back into an ACL SL0 message, which is sent to the agent-client.

2.1.2.3. Messages' Translation

There is implemented a messages' translation in the WSIG. The ACL message is required in the SL0 language, which is the S-expresion. The sequence in paranteses are divided into the first field and the rest. The first field gives a name of the data structure or a function. It is taken as an xml tag name in an xml element. The rest of S-expresion is translated recursively and such structures are placed as content of the xml element.

A SOAP message is represented in the XML format. It consists of well nested xml's elements. An xml's element is translated as following. A name of the element is taken as the first field of an S-expresion. A content of the element is recursively translated and structures are apended as the rest of the S-expresion.

The ACL SL0 format allows the named arguments (slots) in a function's call. A translation process is slightly modified in the rest of an S-expresion. A Slot is taken as one structure insted of a separation into a name and content. It is treated as a new xml's element, which encapsulated a slot's content. A name of the element is the slot's name. An xml's attribute "fipa-attribute" is added to the element with value "true" to indicate the situation. A content of the slot is recursively translated as an S-expresion into the xml's content.

An example of a slots' translation:

```
(store :where box2 :value 20)

<store>
  <where fipa-atribute="true">
    <BO_String>box2</BO_String>
  </where>
  <value fipa-atribute="true">
    <BO_Integer>20</BO_Integer>
  </value>
</store>
```

Attributes of the element are treated as especial ones. If there are no one, then the translation is as above. In case, some attributes exist, the one level of a S-expresion is added. An original content is stored under slot ":xml-element". Attributes are stored as a set of FIPA management properties in ":xml-attributes" slot. A name of the original content is prepended by "xml-tag-" and used as the first field in the S-expresion added.

Such structures are used also in case, when a JADE agent wants to call a WS server with requirement to use xml's attributes.

An example of an xml attributes' translation:

```
  <q id="t12">Foo</q>

  (xml-tag-q
     :xml-element (q Foo)
     :xml-attributes (set ( property :name id :value t12 )))
```

2.1.3. The JADE Agent Server's Example

At first, an agent must provides some services. A service "plus" is implemented in an example. It takes some numbers and performs mathematical operation plus. The service is written in doFIPARequest procedure.

```
...
  private String UNNAMED = "_JADE.UNNAMED";
  private Logger log =
Logger.getLogger(TestAgentServer.class.getName());
  private SLCodec codec = new SLCodec(0);
  public static final String SERVICE_PLUS = "plus";
  private int convId = 0;


  protected void setup() {
    log.info("A TestAgentServer is starting.");

    // add behaviour of the Agent
    this.addBehaviour( new CyclicBehaviour( this ) {
      public void action() {
        ACLMessage msg = myAgent.receive();
        if ( msg != null ) {
          switch ( msg.getPerformative() ) {
            case ACLMessage.REQUEST:
              doFIPARequest( msg );
              break;
            default:
              // other messages are ignored
              break;
          }

          try {
            log.debug("A testAgentServer receives: "
              + SL0Helper.toString(msg) );
          }catch ( Exception e ) {
            log.error(e);
          }

        }else{
          block();
        }
      }
```

```
    } );
...

  /**
   * serves a request
   *
   * @param acl a request
   */
  private void doFIPARequest( ACLMessage acl ) {
    ACLMessage resp = acl.createReply();
    AbsContentElement ac = null;
    AbsObject ao, ao2;
    long sum = 0;
    String str = "";

    // decode the request
    try {
      ac = codec.decode( BasicOntology.getInstance(), acl.getContent()
);
    }catch ( CodecException ce ) {
      str = "(error CodecException ( " + ce + " ))";
      SL0Helper.fillAsNotUnderstood( acl, resp, str );
      send(resp);
      return;
    }

    if( null == ac ) {
      str = "(error action null)";
      SL0Helper.fillAsNotUnderstood( acl, resp, str );
      send(resp);
      return;
    }

    if ( ! SL0Vocabulary.ACTION.equalsIgnoreCase( ac.getTypeName()) ) {
      str = "(unknown action_format)";
      SL0Helper.fillAsNotUnderstood( acl, resp, str );
      send(resp);
      return;
    }else{
      // parse the action
      ao = FIPASL0ToSOAP.getActionSlot( ac );
      if ( null == ao ) {
        str = "(unknown action_slot_format)";
        SL0Helper.fillAsNotUnderstood( acl, resp, str );
        send(resp);
        return;
      }

      // check a service name
      String opName = ao.getTypeName();
      if ( SERVICE_PLUS.equalsIgnoreCase(opName) ) {
        // unnamed parameters are expected
        if ( ! FIPASL0ToSOAP.isWithUnnamed(ao) ) {
          str = "(unknown (format " + opName + " ))";
          SL0Helper.fillAsNotUnderstood( acl, resp, str );
          send(resp);
          return;
        }
```

```
        // do plus on agruments
        String[] name = ao.getNames();
        for(int i = 0; i < ao.getCount(); i ++ ) {
          // get unnamed slot
          ao2 = ao.getAbsObject( UNNAMED+i );
          try{
            sum += ((AbsPrimitive)ao2).getLong();
          }catch(java.lang.ClassCastException cce) {
            str = "(error (argument_format " + opName + " at " + i + "
))";
            SL0Helper.fillAsNotUnderstood( acl, resp, str );
            send(resp);
            return;
          }
        }
        resp = SL0Helper.createInformResult( acl, "" + sum );
      }else{
        str = "(unknown (service " + opName + " ))";
        SL0Helper.fillAsNotUnderstood( acl, resp, str );
      }
    }

    send(resp);
  }
...
```

The agent register the service in the DF. A type slot is set to "web-service". A property type of the service is set too. There is choice, which one uses, but both may be occurred. A registration request is construct as an ACL message. (to call DFService.register(this, dfad) rises problems)

```
...
  private DFAgentDescription dfad = new DFAgentDescription();

  protected void setup() {
...
    // ------------------------------
    // register the agent into the DF

    // pripare a message
    ACLMessage msg = new ACLMessage( ACLMessage.REQUEST );
    AID  dfAID = new AID( "df", AID.ISLOCALNAME );
    msg.addReceiver( dfAID );
    msg.setSender( this.getAID());
    msg.setConversationId( "conv_" + convId ++ );
    msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
    msg.setOntology(FIPAManagementVocabulary.NAME);

    // pripare a DFAgentDescription
    dfad.setName( this.getAID());
    dfad.addLanguages( FIPANames.ContentLanguage.FIPA_SL0 );
    dfad.addProtocols( FIPANames.InteractionProtocol.FIPA_REQUEST );
    ServiceDescription sd;
    sd = new ServiceDescription();
    sd.setName( SERVICE_PLUS ); // here is the service name
    sd.addLanguages( FIPANames.ContentLanguage.FIPA_SL0 );
    sd.addProtocols( FIPANames.InteractionProtocol.FIPA_REQUEST );
```

```
    sd.setType("web-service");
    // or set properties
    Property p = new Property("type","(set web-service)");
    sd.addProperties( p );
    dfad.addServices(sd);

    //set register's argument
    Register reg = new Register();
    reg.setDescription(dfad);

    // create registration's action
    Action action = new Action( this.getAID(), reg );

    // send the request for registration
    try {
      getContentManager().registerLanguage( codec );
      getContentManager().registerOntology(
        FIPAManagementOntology.getInstance());
      getContentManager().fillContent(msg, action);
      send(msg);
    }catch (Exception e) {
      // something is wrong
      e.printStackTrace();
    }

    // ------------------------------
...
```

The example is stored in com.whitestein.wsig.test.TestAgentServer.java file. It is compiled as a default and is occured in a jar file generated. A script `misc/run_TestAgentServer.sh` or `misc\run_TestAgentServer.bat` runs the example. The JADE and the WSIG is already running.

An WS client wants to use a service, which performs mathematical operation plus. At first, the client must do search for an operation plus in WSIG's UDDI repository. An access point, an operation name, and a WSDL's structure are obtained from UDDI. The client constructs a SOAP request message. The message is sent into the access point, which is WSIG's access point. An answer is received by the client from WSIG. A result from the plus operation is stored in the answer.

The example of the WS client is stored in com.whitestein.wsig.test.TestSOAPClient.java file. A script `misc/run_TestSOAPClient.sh` or `misc\run_TestSOAPClient.bat` runs the example.

```
...

  private final static String fipaServiceName = "plus";

  private static Logger log =
    Logger.getLogger( TestSOAPClient.class.getName());

  private UDDIProxy uddiProxy;

  /**
```

```
   * sets up the uddi4j. It starts components required.
   *
   */
  private void setupUDDI4j() {
    // to register into UDDI
    // structures used for a communication with UDDI is retrieved
    Configuration c = Configuration.getInstance();

    uddiProxy = new UDDIProxy();
    synchronized ( c ) {
      // synchronized on main Configuration instance
      // to prevent changes in configuration

      System.setProperty( Configuration.KEY_UDDI4J_LOG_ENABLED,
        c.getUDDI4jLogEnabled());
      System.setProperty( Configuration.KEY_UDDI4J_TRANSPORT_CLASS,
        c.getUDDI4jTransportClass());

      // Select the desired UDDI server node
      try {
        // contact a back end UDDI repository
        uddiProxy.setInquiryURL(c.getQueryManagerURL());
        uddiProxy.setPublishURL(c.getLifeCycleManagerURL());
      }catch( Exception e ) {
        log.error(e);
      }
    }
  }

  /**
   * finds services wanted
   * @return a list of services
   */
  private ServiceList findServices() {
    ServiceList sl = new ServiceList(); // default is an empty list
    try {
      String businessKey = "";   // all business
      Vector names = new Vector(1);
      names.add( new Name("%WSIG%") );  // substring is WSIG

      CategoryBag cb = new CategoryBag();
      KeyedReference kr = new KeyedReference();
      kr.setTModelKey("uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4"); //
uddi-org:general_keywords
      kr.setKeyName("fipaServiceName");
      kr.setKeyValue( fipaServiceName );
      cb.add( kr );
      TModelBag tmb = new TModelBag();  //empty
      FindQualifiers fq = new FindQualifiers();  //empty

      sl = uddiProxy.find_service(
        businessKey,
        names,
        cb,
        tmb,
        fq,
        10 );

    } catch ( UDDIException ue ) {
```

```
      log.debug( ue );
    } catch ( TransportException te ) {
      log.debug( te );
    }
    return sl;
  }

  /**
   * writes out a list of services into a log
   *
   */
  private void writeToLog( ServiceList list ) {
    ServiceInfo info;
    ServiceInfos infos = list.getServiceInfos();
    String s;
    int k;
    for ( k = 0; k < infos.size(); k ++ ) {
      info = infos.get( k );
      s = info.getDefaultNameString();
      log.debug(" a service found: " + s );
    }
  }

  /**
   * performs a test
   */
  private void test(){
    setupUDDI4j();

    // find Services
    ServiceList sList = findServices();
    if ( log.isDebugEnabled() ) {
      writeToLog( sList );
    }
...
```

An AccessPoint, an operation name, and a WSDL namespace is required to call the operation. UDDI records related to service must be traversed to obtain the information.

```
...
    ServiceInfo info;
    ServiceInfos infos = sList.getServiceInfos();

    if ( infos.size() < 1 ) {
      log.info(" No service is available.");
      return;
    }

    info = infos.get( 0 );
    ServiceDetail sd = null;
    try {
      sd = uddiProxy.get_serviceDetail( info.getServiceKey() );
    }catch ( UDDIException ue ) {
      log.debug( ue );
    }catch ( TransportException te ) {
      log.debug( te );
    }
    if ( null == sd ) {
```

```
      log.info(" No service is available in the 2nd step.");
      return;
    }

    Vector sv = sd.getBusinessServiceVector();
    if ( sv.size() < 1 ) {
      log.info(" No service is available in the 2nd step.");
      return;
    }

    // take the first service
    BusinessService bs = (BusinessService) sv.elementAt( 0 );

    // get an accessPoint
    BindingTemplates bts = bs.getBindingTemplates();
    if ( bts.size() < 1 ) {
      log.info(" No bindingTemplate is available. ");
      return;
    }
    BindingTemplate bt = bts.get(0);
    AccessPoint aPoint = bt.getAccessPoint();
    URL ap = null;
    try {
      log.info(" An accessPoint is " + aPoint.getText()
        + " and type is " + aPoint.getURLType() );
      ap = new URL( aPoint.getText() );
    }catch (MalformedURLException mfe) {
      log.error( mfe );
      return;
    }

    // get TModel, only one is expected
    TModelInstanceDetails tmids = bt.getTModelInstanceDetails();
    if ( tmids.size() < 1 ) {
      log.info(" No TModelInstanceInfo is available. ");
      return;
    }
    TModelInstanceInfo tmii = tmids.get(0);
    String tmk = tmii.getTModelKey();
    TModelDetail tmd = null;
    try {
      tmd = uddiProxy.get_tModelDetail( tmk );
    }catch ( UDDIException ue ) {
      log.debug( ue );
    }catch ( TransportException te ) {
      log.debug( te );
    }

    if ( null == tmd ) {
      log.info(" No TModelDetail is available.");
      return;
    }

    Vector tmdv = tmd.getTModelVector();

    if ( tmdv.size() < 1 ) {
      log.info(" No TModel is available.");
      return;
    }
```

```
    TModel tm = (TModel) tmdv.get(0);

    // get wsdl url from TModel, only one is expected
    OverviewDoc ovd = tm.getOverviewDoc();
    if ( null == ovd ) {
      log.info(" No OverviewDoc is available in TModel.");
      return;
    }
    String wsdlURL = ovd.getOverviewURLString();
    if ( null == wsdlURL ) {
      log.info(" OverviewDoc's URL is null.");
      return;
    }
    log.info(" TModel refers to wsdl: " + wsdlURL );

    // get an operation for fipaServiceName
    CategoryBag cb = bs.getCategoryBag();
    KeyedReference kr;
    int k;
    for ( k = 0; k < cb.size(); k ++ ) {
      kr = cb.get( k );

      if ( fipaServiceName.equalsIgnoreCase( kr.getKeyName() )) {

        // it is found, call it
        callOperation( ap, kr.getKeyValue(), wsdlURL );
          // it is better to extract nameSpace from the wsdl in the
future
        return;
      }
    }
  }
...
```

A SOAP request is constructed in generatePlus method. A call to accessPoint is invoked then.

```
...
  /**
   * calls an operation
   *
   * @param accessPoint access point of a WS
   * @param opName an operation's name
   * @param wsdlNS a wsdl name space
   */
  private void callOperation( URL accessPoint, String opName, String
wsdlNS ){
    URL serverURL = accessPoint;
    HttpURLConnection c = null;

    SOAPMessage retSOAP = null;

    // generate a test's message
    String str;
    int[] values = { 3, 5, 7 };
    str = generatePlus( opName, values, wsdlNS );

    SOAPMessage soap;
```

```
    soap = new Message( str, false,
      "application/soap+xml; charset=\"utf-8\"", "" );

    // debug to write down
    ByteArrayOutputStream baos;
    try {
      baos = new ByteArrayOutputStream();
      soap.writeTo(baos);
      log.info("A SOAP sent: \n  " + baos.toString());
    } catch (SOAPException e) {
      log.error(e);
    } catch (IOException ioe) {
      log.error(ioe);
    }

    try {
      // send a request
      c = WSEndPoint.sendHTTPRequest( serverURL, soap );

      // read a response
      retSOAP = WSEndPoint.receiveHTTPResponse( c );

      // debug to write down
      if ( retSOAP != null ) {
        try {
          baos = new ByteArrayOutputStream();
          retSOAP.writeTo(baos);
          log.info("A SOAP received: \n  " + baos.toString());
        } catch (SOAPException e) {
          log.error(e);
        } catch (IOException ioe) {
          log.error(ioe);
        }
      }else {
        log.info("A SOAP received: null.");
      }


      // release resources
      c.disconnect();

    }catch (SOAPException se) {
      log.error(se);
    }catch (IOException ioe) {
      log.error(ioe);
    }finally{
      if (c != null) {
        c.disconnect();
      }
      isRunning = false;
      //return;
    }

  }

  /**
   * generates a SOAP message.
   *
   * @param op_name a name of a operation
```

```
   * @param nums an array of integers as arguments
   * @param wsdlNS a wsdl namespace
   * @return a message in string
   */
  public static String generatePlus( String op_name, int[] nums, String
wsdlNS ) {
    String str =
      "<?xml version=\"1.0\" encoding=\"UTF-8\"?> " +
      " <soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">  " +
      "  <soapenv:Body>\r\n" +
      "   <tns:" + op_name + " xmlns:tns=\"" + wsdlNS + "\" >   ";
    for ( int k = 0; k < nums.length; k ++ ) {
      str += "    <tns:BO_Integer>" + nums[k] + "</tns:BO_Integer>\r\n";
    }
    str +=
      "   </tns:" + op_name + ">  " +
      "  </soapenv:Body>   " +
      " </soapenv:Envelope>\r\n";
    return str;
  }

...
```

2.1.4. The Web Services Server's Example

The WS server must provide some operation. A simple operation "echo" returns an echo string. It is implemented in "doRequest" method of `com.whitestein.wsig.test.TestSOAPServerConnection.java`. A name of the operation is checked. The SOAP with echo string is sent back to a client called the service. An fault SOAP is generated, when the name is not good.

```
...
  public static final String OP_1 = "echo";
...
  private void doRequest( WSMessage wsMsg, OutputStream os ) throws
IOException {

    String answer = "";
    String opName =
wsMsg.getTheFirstUDDIOperationId().getWSDLOperation();
    if ( OP_1.equals( opName )) {
      answer =
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?> " +
        "       <soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">  " +
        "        <soapenv:Body> " +
        "<tns:String xmlns:tns=\""+ wsdlTargetNamespace +"\" >" +
        "Echo string" +
        "</tns:String>   " +
        "        </soapenv:Body>   " +
        "       </soapenv:Envelope>\r\n";
```

```
    }else {
      answer =
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?> " +
        "       <soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">  " +
        "        <soapenv:Body> " +
        "<soapenv:Fault>" +
        "<soapenv:faultcode>soapenv:Client</soapenv:faultcode>" +
        "<soapenv:faultstring>Unknown operation.</soapenv:faultstring>"
+
        "<soapenv:faultactor></soapenv:faultactor>" +
        "</soapenv:Fault>" +
        "        </soapenv:Body>   " +
        "       </soapenv:Envelope>\r\n";
    }

    byte[] content = answer.getBytes("UTF-8");
    Connection.sendBackSOAPContent( content, os );
  }
...
```

The server must register itself into the WSIG after it is started. The WSIG provides a UDDI registration interface as a valid UDDI repository. All requests for the WSIG are sent into back-end UDDI repository. From these requests relevant information are extracted and used by the WSIG. A code of registration is in `com.whitestein.wsig.test.TestSOAPServer.java` in register() method. A businessKey is used from the WSIG's configuration. The server becoms a servise of the same business like the WSIG.

The code of the SOAP server is in `com.whitestein.wsig.test.TestSOAPServer.java` and `com.whitestein.wsig.test.TestSOAPServerConnection.java` files.

The JADE agent wants to use an operation, which return some string. It must know the name, the "echo". The better categorization makes it possible to do search for that operation withou a special knowledge of service name in the future. The agent creates a search request into DF. The WSIG provides such information in "web-service.operation" property of service.

```
...
  private void doSearch() {
     DFAgentDescription template = new DFAgentDescription();
     ServiceDescription sd = new ServiceDescription();
     Property p = new Property(
       Configuration.WEB_SERVICE + ".operation",
       wsdlOperation );
     sd.addProperties( p );
     template.addServices( sd );
     try {
       DFAgentDescription[] res = DFService.search( this, template);
...
```

In success, the agent uses an operation found. The behavour is added to wait for a response. The request is constructed and sent.

```
...
        serviceName = findServiceName( res[0] );
        if ( null == serviceName ) {
          log.info( "No service is found." );
          doDelete();
          return;
        }

        this.addBehaviour( new CyclicBehaviour( this ) {
          public void action() {
            ACLMessage msg = myAgent.receive();
            if ( msg != null ) {
              processResponse( msg );
            }else {
              block();
            }
          }
        });

        final ACLMessage m = createRequest( aid, serviceName );
        this.addBehaviour( new OneShotBehaviour( this ) {
          public void action() {
            send( m );
          }
...
```

The request for the operation is created in the ACL SL0 format. A content is filled as action where the operation takes a place of the action part.

```
...
  private synchronized ACLMessage createRequest( AID aid, String service
) {
    ACLMessage msg = new ACLMessage( ACLMessage.REQUEST );
    msg.addReceiver( aid );
    msg.setSender( this.getAID());
    msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
    msg.setConversationId( "conv_" + convId ++ );
    msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
    msg.setOntology("AnOntology");
    msg.setContent(
      "((action                \n" +
      "    (agent-identifier                          \n" +
      "      :name " + Configuration.getInstance().getGatewayAID() + " )
\n" +
      "               (" + service + "          \n" +
      "   ) ))");
    return msg;
  }
...
```

2.1.4.1. The JADE client Example 2

The example is related to WS server already running. The server must provides the access point and the WSDL information. A registration must be performed by another peace of a code. A code of a registrator for the Google is in `com.whitestein.wsig.test.TestGoogleRegistration.java` and `com.whitestein.wsig.test.TestGoogleRegistrationConnection.java` files. The server must provide WSDL file because Google does not have one's public accessible on some HTTP server. The UDDI require a WSDL to be accessible.

A JADE agent client is described now. At first, the agent finds a service, which is translation of the operation "doGoogleSearch". Let the service name be "operation1". Arguments for operation must be consulted with WSDL file. An element key is required to have a value of the access key, which is obtained from Google after filling a registration. The Googles key must be stored in the wsig.properties file then. A query string is stored in "q" element. It is a string "Foo" in the example. The WS server requires an element attribute "xsi:type" for every data element. It is stored in a fipa slot ":xml-attributes" as is described in the translation of xml attributes. Then the TestAgentWithArgs agent construct a message as following:

```
(operation1
  (xml-tag-key
    :xml-element (key ABcdABcdABcdABcdABcdABcdABcd123)
    :xml-attributes
      (set ( property :name xsi:type :value xsd:string )))
  (xml-tag-q
     :xml-element (q Foo)
     :xml-attributes (set ( property :name xsi:type :value xsd:string
)))
  (xml-tag-start
    :xml-element (start 0)
    :xml-attributes (set ( property :name xsi:type :value xsd:int )))
  (xml-tag-maxResults
    :xml-element (maxResults 4)
    :xml-attributes (set ( property :name xsi:type :value xsd:int )))
  (xml-tag-filter
    :xml-element (filter true)
    :xml-attributes (set ( property :name xsi:type :value xsd:boolean)))
  (xml-tag-restrict
    :xml-element (restrict)
    :xml-attributes (set ( property :name xsi:type :value xsd:string )))
  (xml-tag-safeSearch
    :xml-element (safeSearch false)
    :xml-attributes (set ( property :name xsi:type :value xsd:boolean
)))
  (xml-tag-lr
    :xml-element (lr)
    :xml-attributes (set ( property :name xsi:type :value xsd:string )))
  (xml-tag-ie
    :xml-element (ie latin1)
    :xml-attributes (set ( property :name xsi:type :value xsd:string )))
  (xml-tag-oe
    :xml-element (oe latin1)
    :xml-attributes (set ( property :name xsi:type :value xsd:string )))
```

)

A code of the agent with arguments is in `com.whitestein.wsig.test.TestAgentWithArgs.java`. It expected two arguments: the first is an original WSDL operation name and the second is a string of arguments of an operation's call, where the string is in format suitable to put direct into SL0. (for "(plus 1 2 3)" is string of arguments "1 2 3") The WSDL operation name is used to search for a service name.

The Google registration may be run by a script misc/run_TestGoogle.sh or misc\run_TestGoogle.bat. The agent invocation may be done through the WSIG's GUI.

3. PDFs