Agreement Document Analyser

User Guide

Contenido

What's ADA?

ADA is a tool to analyse Service Level Agreements (SLAs) specified with WS-Agreement recommendation. What's a SLA? It's is a part of a service contract where the level of service is formally defined. And what's WS-Agreement? It's a protocol specification for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language.

What can ADA do for me? ADA is able to detect if an agreement document has errors, and also if an agreement is possible between two parties from documents of each party

ADA flavors

You can use ADA by four ways. We provide a web front-end to final users, and three ways to integrate your own application with ADA: a SOAP/WSDL web service, a set of OSGi bundles, and a java standalone version.

- **ADA web front-end**: is the current front end for final users.
- **ADA Web Service**: we provide a WSDL with most of ADA operations. You can consume it for your application.
- **ADA OSGi**: ADA is also available as a set of OSGi bundles. OSGi (www.osgi.org) is a java specification for services integration.
- **ADA standalone**: moreover, ADA is available as an extensible library.

Installation

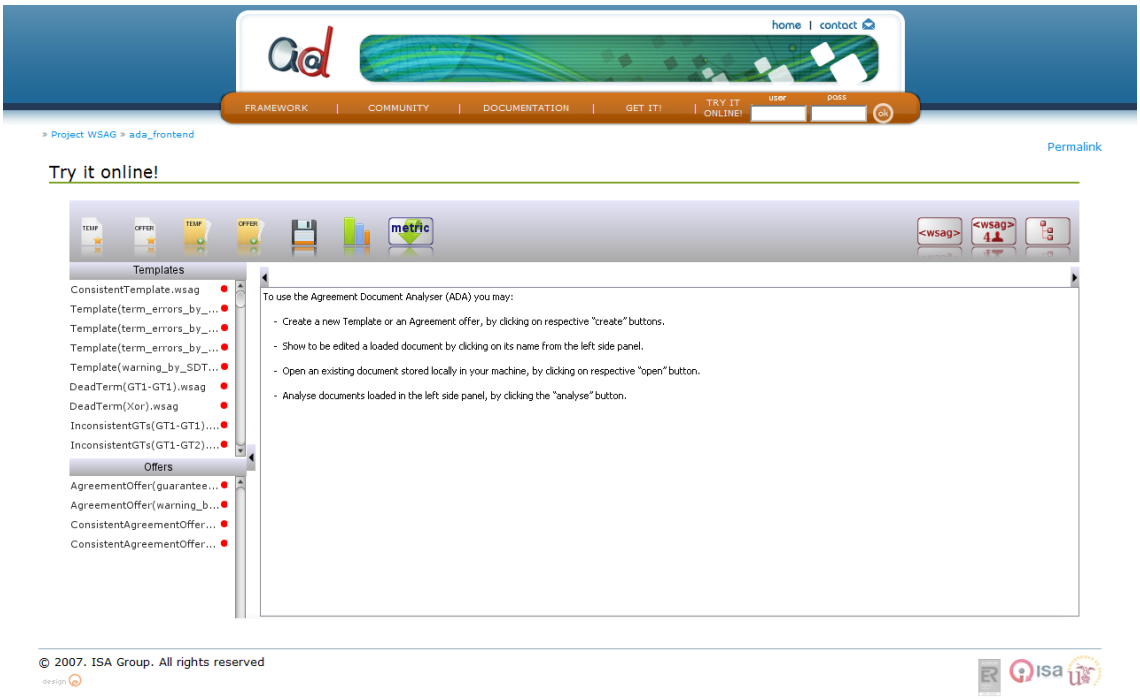
Web front-end version

Web front-end has been developed with Java FX technology. Java FX is a technology to create rich internet applications, but compiled on java code. Final users only should to install java desktop version.

Unfortunately, some browsers have problems to work correctly with this technology. In the next table, we present compatibilities over most common browsers.

OS	Brower	Behavior	Comments
Windows	IE 8	Ok	
	Firefox 3	Ok	
	Chrome	Problems	Documents are not shown
Mac	Safari	Problems	JavaFX usually doesn't load
Ubuntu	Firefox	Ok	

ADA front-end is available at www.isa.us.es/ADA, *try it!* section. If you can see the screen below these lines, you are ready to start with ADA front-end:



Web service version

Steps:

1. WSDL address is http://150.214.188.147:8081/ADAService?wsdl. Have a quick look, and make sure the site is online.
2. Use a web service tool, as Apache Axis for java, or an IDE that provides WSDL tools, as Eclipse for java or Visual Studio for .NET. These tools can create stub classes to use a web service. If you are using Eclipse:
    a. Install (if don't) Axis tools for eclipse. Open eclipse/ *Help/ Install new software*. On "Working with", select the name of your eclipse distribution, and when screen loads all plugins, go to *Web, XML and Java EE Development/Axis2* Tools and select it.
    b. Create an empty project
    c. File/ New/ Other/ Web Services/ Web Service Client. Paste WSDL location on the upper box, and select assemble (second level) on the left. Next and finish. Stub classes and imported libraries are now on the project.

OSGi version

Steps:

1. Get an OSGi distribution. There are several OSGi distributions. We recommend you Equinox (integrated with eclipse) or Felix. Both are the most stable and mature implementations of OSGi standard. If you want do OSGi subsection of QuickStart, you should use Equinox.

1.1. If you want to use ADA through Equinox on Eclipse:
    a. Get an eclipse distribution at www.eclipse.org if you don't have it. If you don't know what version to download, we recommend you Eclipse classic. You have enough information about installing and using eclipse on this site.

b. When installed, copy all ADA jars (on root directory and lib folder) to "dropins" eclipse folder.
c. Launch eclipse, and open Plug-in registry view (Window/ Show view/ Other/ Plug-in development/ Plug-in registry). Make sure jars you have copied on previously step are installed and started. If they aren't started, right click on it, "Show advanced operations". Then, right click again, and "Start".

1.2. If you want to use through Felix:
a. Go to http://felix.apache.org and download last Felix version (Felix Framework Distribution). If you already have a Felix distribution, make sure you delete "felix-cache" folder to avoid problems.
b. Decompress it to a folder.
c. Enter on Felix folder, and copy ADA jars to "bundle" folder.
d. Start Felix from command line: enter on Felix folder, and type "java –jar bin/felix.jar". You should see a new prompt.
e. Type "lb" to check ADA bundles are installed and started.

Standalone version

ADA is also available as a standalone library. For this format, you need to import ADA, antlr and choco libraries on your project, and copy file ADAConfig.xml to project root folder. Moreover, you need to locate any document or metric you are going to use.

Quick start

We are going to working with the same example for each way of use.

Web front-end version

Steps:

1. Enter on www.isa.us.es/ada, and go to *TRY IT ONLINE!* section.
2. If you have followed installation section, you have to see ADA front-end main menu when JavaFx loading finishes.
3. Click on three bars icon  , and a new menu will appear with available operations. Select *check conflicts*, and accept.
4. Now, you have to select a document for the operation. Select consistent template, click add, and then execute.
5. During the analysis, messages about consistency and warnings will appear. Result should be a consistent document, but with dead terms. Close analysis advices.
6. Next, click again on three bars icon, and select now *check compliance conflicts*.
7. This time, you have to select two documents: a template and an offer. Select the same template than before, ConsistentTemplate, and select *ConsistentAgreementOfferNonCompliant…* as offer. Then, execute. Result should be non compliance.

Web service version

Once you have created stub classes from ADA wsdl (http://150.214.188.147:8081/ADAService?wsdl), you can use a code like the figure shows. Remember put it into a main method:

```
//a method to extract bytes from a file
ADAServiceV2PortType ada = new
      ADAServiceV2PortTypeProxy("http://150.214.188.147:8081/ADAService");
byte[] template = getBytesFromFile(new File("ConsistentTemplate.wsag"));
try {
   boolean res = ada.checkDocumentConsistency(template);
   if (res){
      //if template is consistent
      byte[] offer = getBytesFromFile(new
File("ConsistentAgreementOffer.wsag"));
      boolean compliant = ada.isCompliant(template,offer);
      System.out.println("Is the offer compliant? "+compliant);
   }
} catch (RemoteException e) {
   e.printStackTrace();
}
```

You can get ConsistentTemplate and AgreementOffer files at https://forja.rediris.es/docman/view.php/606/1295/ConsistentTemplate.wsag and https://forja.rediris.es/docman/view.php/606/1296/ConsistentAgreementOffer.wsag

We provide you method getBytesFromFile:

```
private static byte[] getBytesFromFile(File f) {
       int size = (int) f.length();
       byte[] res = new byte[size];
       InputStream in;
       try {
              in = new FileInputStream(f);
              in.read(res);
       } catch (FileNotFoundException e) {
              e.printStackTrace();
       } catch (IOException e) {
              e.printStackTrace();
       }
       return res;
}
```

Copy code, files, and execute the main class.

Output should be:

```
Is the offer compliant? false
```

OSGi version

Steps:

1. Start eclipse, and create a new Plug-in project: File/ New/ Other/ Plug-in Development/ Plug-in Project. On "This plug-in is targeted to run with", select Equinox, and click next. Click next, and finish.
2. When project has been created, go to the MANIFEST file, and on "Dependencies", add a required plug-in: es.us.isa.ADACore. Save it.

3. Now, go to the Activator class. On start method, you should get services you want (ADA on our case), and on stop method, you should release these services (you should save service references you get). Your start method may look as follows.

```
public void start(BundleContext context) throws Exception {
       //adaReference is an attribute you have to declare
      //type: ServiceReference
      adaReference =
          context.getServiceReference(ADA.class.getCanonicalName());
       ADA ada = (ADA) context.getService(adaReference);
       if (ada != null){
             System.out.println("ADA recovered");
             //use ada like on standalone version
             useADA(ada);
       }
}

public void useADA(ADA ada){
AbstractDocument doc = ada.loadDocument("ConsistentTemplate.wsag");
ConsistencyOperation op =
     (ConsistencyOperation)ada.createOperation("Consistency");
op.addDocument(doc);
ada.analyze(op);
boolean isConsistent = op.isConsistent();
System.out.println("Is the template consistent?"+isConsistent);
if (isConsistent){
   AbstractDocument offer = ada.loadDocument("NonCompliantOffer.wsag");
   ComplianceOperation op2 = (ComplianceOperation)
            ada.createOperation("compliance");
   //first a template, later an offer
   op2.addDocument(doc);
   op2.addDocument(offer);
   ada.analyze(op2);
   boolean compliant = op2.isCompliant();
   System.out.println("Is the offer compliant? "+compliant);
}

}
```

You can get ConsistentTemplate and AgreementOffer files at https://forja.rediris.es/docman/view.php/606/1295/ConsistentTemplate.wsag and https://forja.rediris.es/docman/view.php/606/1296/ConsistentAgreementOffer.wsag. To load models, you have to copy them into your eclipse folder.

And your stop method:

```
public void stop(BundleContext context) throws Exception {
     if (sr != null){
         context.ungetService(adaReference);
     }
}
```

4. Finally, run the project as an OSGi application. Select "Run configurations", and create a new OSGi Framework configuration (double click). For workspace bundles, select only this project, and for Target platform, unselect all (click on Target platform box), and then click on "Add requires bundles". Then select every ADA bundle (all es.us.isa packages), put them priority 1, apply and run.

Standalone version

If you have followed steps pointed on installation section, you only have to execute a Main class with the next code:

```
ADA ada = new ADA();
AbstractDocument doc = ada.loadDocument("ConsistentTemplate.wsag");
ConsistencyOperation op =
     (ConsistencyOperation)ada.createOperation("Consistency");
op.addDocument(doc);
ada.analyze(op);
boolean isConsistent = op.isConsistent();
System.out.println("Is the template consistent?"+isConsistent);
if (isConsistent){
   AbstractDocument offer = ada.loadDocument("NonCompliantOffer.wsag");
   ComplianceOperation op2 = (ComplianceOperation)
            ada.createOperation("Compliance");
   //first a template, later an offer
   op2.addDocument(doc);
   op2.addDocument(offer);
   ada.analyze(op2);
   boolean compliant = op2.isCompliant();
   System.out.println("Is the offer compliant? "+compliant);
}
```

Deeping ADA

Background

In recent years, service level agreements (SLAs) have gained great importance specially in the context of service-oriented computing since they play a major role to regulate both functional and non-functional properties regarding the provisioning of a service. A proof of that importance is the rise of standards to define the characteristics of SLAs and how they must be created. The most significant representative of them is WS–Agreement, a proposed recommendation of the Open Grid Forum (OGF) that provides a XML schema for defining SLAs and a protocol for creating them based on templates and agreement offers. Specifically, the schema establishes several terms types to describe the agreed service, in other words the functional properties; and some guarantees on it, in other words nonfunctional properties. In turn, the protocol allows publishing a template informing about the functional and non-functional capabilities of a party. Then, third-parties would send their agreement offers to the template publisher in order to try to achieve an agreement and the template publisher will decide.

Since the sign of a conflicting SLA may involve financial crashes to the involved parties, ADA is developed with the aim of handling right SLAs. Thus, ADA analyses WS–Agreement documents to detect and explain several kinds of conflicts that may appear: (1) between inconsistent terms of an unique document, either in a template or an agreement offer; and (2) between an template and an agreement offer received from a third-party.

Model

WS-Ag model is specified by the Open Grid Forum (OGF). The specification is available on http://www.ogf.org/documents/GFD.107.pdf. ADA uses this specification, and extends it.

Document types

- Template: you can use a template to expose services that you provide, or that you need. Templates have three sections: context information, terms and creation constraints.
- Offer: an offer is a response to a template that concretes terms defined on it. Offers only have two sections: context information and terms.

Document elements

We can structure document elements on three categories: context information, terms and creation constraints. At this subsection, we provide examples about each one of the elements on xml syntax.

Context information provides textual information about providers, agreement parties, the agreement's lifetime, or other kind of data of informative value.

```
<wsag:Context>
       <wsag:AgreementInitiator>INeedTranslationCorp.</wsag:AgreementInitiator>
       <wsag:AgreementResponder>ITranslate</wsag:AgreementResponder>
       <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
       <wsag:ExpirationTime>2012-01-01T00:00:00</wsag:ExpirationTime>
       <wsag:TemplateId>1.1</wsag:TemplateId>
       <wsag:TemplateName>TranslateIt!</wsag:TemplateName>
</wsag:Context>
```

Example of agreement context in an offer

Terms describe requested services to be provided. Every term should have a name. We find four types of terms:

- **Service properties**: declaration of those service features that are relevant to the agreement. It is like a declaration of variables, each with a name and a domain.

```
<wsag:ServiceProperties wsag:Name="SP1"
              wsag:ServiceName="TranslationService1">
      <wsag:VariableSet>
              <wsag:Variable wsag:Name="DemandedTranslationTime"
       wsag:Metric="metrics/metric2575304161886105:Time">
                 <wsag:Location>\\DemandedTranslationTime</wsag:Location>
              </wsag:Variable>
       </wsag:VariableSet>
</wsag:ServiceProperties>
```

Example of service properties

- **Service description term**: default value assignment for variables declared on service properties.

```
<wsag:ServiceDescriptionTerm wsag:Name="SDT1"
              wsag:ServiceName="TranslationService1">
       <OfferItem name="Size">20</OfferItem>
       <OfferItem name="HumanSupervised">1</OfferItem>
       <OfferItem name="DemandedTranslationTime">2</OfferItem>
       <OfferItem name="Cost">10</OfferItem> <!-- (20*1)/2 in euros -->
</wsag:ServiceDescriptionTerm>
```

Example of a service description term

- **Guarantee term**
  - Service level objective
  - Qualify condition
  - Business value list

```
<wsag:GuaranteeTerm wsag:Name="TranslationTimeOffer"
                          wsag:Obligated="ServiceProvider">
      <wsag:ServiceLevelObjective>
            <wsag:CustomServiceLevel>
                  DemandedTranslationTime&lt;=100 AND
                  DemandedTranslationTime&gt;0
            </wsag:CustomServiceLevel>
      </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
```

Example of a guarantee term(without qualifying condition)

- **Term compositor**: aggregates several terms. Depending of composition type, one or more terms should been fulfilled.
  - All: every term has to be satisfied
  - Or: one or more terms have to be satisfied
  - Xor: one (and only only) term has to be satisfied

```
<wsag:Terms wsag:Name="TranslationService">
      <wsag:All>
            <wsag:ServiceProperties wsag:Name="SP1"
                  wsag:ServiceName="TranslationService1">
                …
            </wsag:ServiceProperties>
            <wsag:ServiceDescriptionTerm wsag:Name="SDT1"
                  wsag:ServiceName="TranslationService1">
                …
            </wsag:ServiceDescriptionTerm>
                …
                …
       </wsag:All>
</wsag:Terms>
```

Example of a term compositor (without qualifying condition)

Creation constraints section is a section of templates, where you can define constraints over elements.

- **Item:** items are used to restrict elements domain

```
<wsag:Item wsag:Name="Cost">
      <wsag:Location>\\Cost</wsag:Location>
      <wsag:ItemConstraint>
            <xs:restriction base="xs:integer">
                  <xs:minInclusive value="1" />
                  <xs:maxInclusive value="100" />
            </xs:restriction>
      </wsag:ItemConstraint>
</wsag:Item>
```

Example of an item

- **Creation constraint**: creation constraints are used to constraints that may involve several elements, like the sample shows.

```
<wsag:Constraint>
      <Name>CostCreationConstraint</Name>
      <Content>Cost=(Size*HumanSupervised)/DemandedTranslationTime</Content>
</wsag:Constraint>
```

Example of creation constraint

Input file formats

ADA accepts two input file formats for WS-Ag: an xml format, based on WS-Ag specification, and a plain-text format, more friendly and human-readable.

XML format

ADA xml format is based on WS-Ag specification. We provide you 2 examples on this syntax: a template (https://forja.rediris.es/docman/view.php/606/1295/ConsistentTemplate.wsag) and an offer (https://forja.rediris.es/docman/view.php/606/1296/ConsistentAgreementOffer.wsag). You can view more examples at ADA web front-end (www.isa.us.es/ada)

At present, our xml format supports:

- Integer domains, defined on metric files
- Variable declaration on service description terms
- Service description terms, based on pairs attribute-value
- Term compositors (OR, XOR, ALL)
- Guarantee Terms with Qualify Condition and custom Service Level.

Plain text format

WS-Ag4people is a plain-text format to define WS-Agreement documents. It is more human-readable than xml format, and friendlier to edit.

We provide you two examples on wsag4people syntax: a template (https://forja.rediris.es/docman/view.php/606/1297/ConsistentTemplate.wsag4people) and an offer (https://forja.rediris.es/docman/view.php/606/1298/ConsistentAgreementOffer.wsag4people). Also, every sample on xml syntax at front-end can be read as 4people. Just click on wsag4people button when you have opened a document in xml syntax.

Metrics

Metric files are used to define domains for document variables. You can use them on Service Properties or on Service Description Terms when declaring variables. An example is available at https://forja.rediris.es/docman/view.php/606/1299/metricXML.xml. Syntax is as follows:

```
<metricXML>
  <TypeName1 type="integer" min="Min1" max="Max1"/>
```

```
  ...
  <TypeNameN type="integer" min="MinN" max ="MaxN"/>
</metricXML>
```

Operations

Consistency

- Description: checks consistency of an agreement document
- Input parameters: none
- Output parameters: model consistency (Boolean)
- Front-end operation: `check conflicts`
- API usage:
  - Id: `Consistency`
  - Interface: `ConsistencyOperation`
  - Methods:

Method	Comments
`isConsistent(): boolean`	Returns if model is consistent or not

- Web service usage: `checkDocumentConsistency(byte[] arg0): boolean`

Explain inconsistencies

- Description: if an agreement document is not consistency, it looks for explanations
- Input parameters: none
- Output parameters: inconsistency explanations (Set of explanations)
- Front-end name: `explain conflicts`
- API usage:
  - Id: `ExplainNonConsistency`
  - Interface: `ExplainNoConsistencyOperation`
  - Methods:

Method	Comments
`explainErrors(): Map<AgreementElement, Collection<AgreementElement>>`	Returns inconsistency explanations

- Web service usage: `explainInconsistencies(byte[] arg0): AgreementElement2ArrayOfAgreementElementMapEntry[]`

Dead terms

- Description: checks if document has dead terms, or in other words, if any term of the document can be never reached.
- Input parameters: none
- Output parameters: model dead terms (Set of terms)
- Front-end operation: `check conflicts`
- API usage:

- Id: `DeadTerms`
- Interface: `DeadTermsOperation`
- Methods:

Method	Comments
`hasDeadTerms(): boolean`	Returns if model has or not dead terms
`getDeadTerms(): Collection<Term>`	Returns document's dead terms

- Web service usage: `getDeadTerms(byte[] arg0): Term[]`

Explain dead terms

- Description: determines minimal causes of document dead terms.
- Input parameters: dead terms
- Output parameters: dead terms explaining (Function(Term=>Set<Element>))
- Front-end operation: `explain conflicts`
- API usage:
  - Id: `ExplainDeadTerms`
  - Interface: `ExplainDeadTerms`
  - Methods:

Method	Comments
`setDeadTerms(Collection<Term> c): void`	Sets dead terms
`explainDeadTerms():Map<Term, Collection<AgeementElement>>`	Returns document's dead terms causes

- Web service usage: `explainDeadTerms(byte[] arg0, Term[] arg1): Term2ArrayOfAgreementElementMapEntry[]`

Ludicrous terms

- Description: checks if document has ludicrous terms, or in other words, if any term of the document can be never reached.
- Input parameters: none
- Output parameters: model ludicrous terms (Set of terms)
- Front-end name: `check conflicts`
- API usage:
  - Id: `LudicrousTerms`
  - Interface: `LudicrousTermsOperation`
  - Methods:

Method	Comments
`hasLudicrousTerms(): boolean`	Returns if model has or not ludicrous terms
`getLudicrousTerms(): Collection<Term>`	Returns ludicrous terms

- Web service usage: `getLudicrousTerms(byte[] arg0): Term[]`

Explain ludicrous terms

- Description: determines minimal causes of document ludicrous terms.
- Input parameters: ludicrous terms
- Output parameters: ludicrous terms explaining (Function(Term=>Set<Element>))
- Front-end operation: `explain conflicts`
- API usage:
  - Id: `ExplainLudicrousTerms`
  - Interface: `ExplainLudicrousTerms`
  - Methods:

Method	Comments
`setLudicrousTerms(Collection<Term> c): void`	Sets ludicrous terms
`explainLudicrousTerms():Map<Term, Collection<AgeementElement>>`	Returns document's ludicrous terms causes

- Web service usage: `explainLudicrousTerms(byte[] arg0, Term[] arg1): Term2ArrayOfAgreementElementMapEntry[]`

Compliance

- Description: checks consistency of an agreement document
- Input parameters: none
- Output parameters: model consistency (Boolean)
- Front-end usage: `valid`
- API usage:
  - Id: `valid`
  - Interface: `ValidQuestion`
  - Methods:

Method	Comments
`isValid(): boolean`	Returns if model is valid or not

- Web service usage: `isCompliant(byte[] arg0, byte[] arg1): boolean`

Explain non compliance

- Description: checks consistency of an agreement document
- Input parameters: none
- Output parameters: model consistency (Boolean)
- Front-end usage: `valid`
- API usage:
  - Id: `valid`
  - Interface: `ValidQuestion`
  - Methods:

Method	Comments
`isValid(): boolean`	Returns if model is valid or not

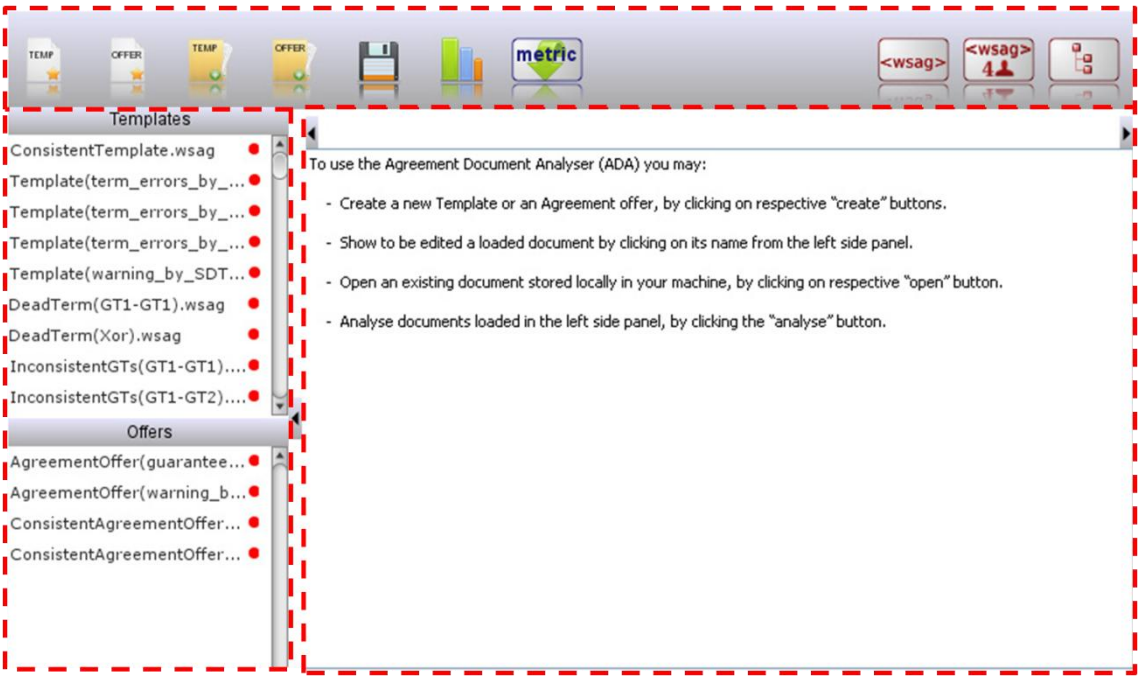
- Web service usage: `explainNonCompliance(byte[] arg0, byte[] arg1): AgreementElement2ArrayOfAgreementElementMapEntry[]`

Analyzers

ADA architecture provides the option to use several analyzers, and choose the best for each case. At now, only one analyzer is available, based on a CSP library (Choco solver), but we are working to widen this area.

ADA web front-end

Quick look



On the image, we can distinguish 3 different places. On left, we have 2 sets of predefined documents: offers and templates. On right, we see document's content when we click on anyone on left. On top many options are available. From the left to right:

- New template: creates a new agreement template
- New offer: creates a new agreement offer
- Upload local template: uploads a template from the local file system
- Upload local offer: uploads an offer from the local file system
- Save to a local file: saves the current document to the local file system
- Analysis operations: opens a new menu to choose an analysis operation
- Upload a metric file (not available at now)
- XML view: shows the current document on xml format
- WS-Ag4people view: shows the current document on wsag4people format
- Tree view: shows the current document on java tree format

Editing and using your documents

You can use default documents that are on the front-end. But you also can create your own documents. If you create it from the front-end, it will load a default skeleton with most common options. You can upload previously edited documents too, and analyze them. However, at now you need to reference default front-end metric file

Document views

- WS-Agreement xml view: this is the default view. You can read documents at standard xml notation.
- WS-Ag 4 people view: using this option, you can view xml documents on 4people syntax, and edit them.
- Tree view: this is a schematic view, to check document structure.

Using operations

Operations menu opens on  icon. You can choose over four options:

- **Check conflicts**: works over one document, and combines consistency, dead terms and ludicrous terms operations. If the document is consistent, it looks for dead/ludicrous terms. In another case, it reports document inconsistency.
- **Explain conflicts**: goes further than check conflicts. If the document is inconsistent, it shows those set of terms that cause the inconsistency. If the document is consistent but it has dead/ludicrous terms, it shows set of terms that cause the issues. Terms painted with the same color collide between them.
- **Check compliance conflicts**: works over two documents, a template and an offer. It checks if the offer is compliant with the template. But first, it checks that both documents are consistent. This is a complex operation, and it may take a while.
- **Explain non-compliance conflicts**: looks for causes of non compliant. It may take a long time. It paints elements that cause compliance conflicts on offer and template. Terms painted with the same color collide between them.

ADA standalone

Basics

ADA has a facade class, named ADA. User should interact with ADA through this class. It provides methods to work with models and operations.

ADA usage through standalone form has 6 basic steps.

```
//1. Instantiate ADA
ADA ada = new ADA();

//2. Load a document
AbstractDocument doc = ada.loadDocument("Template.wsag");

//3. Create operation
ConsistencyOperation op =
         (ConsistencyOperation)ada.createOperation("Consistency");
```

```
//4. Add the document to the operation
op.addDocument(doc);

//5. Analyze operation
ada.analyze(op);

//6. Recover the result
boolean isConsistent = op.isConsistent();
System.out.println("Is the document consistent?"+isConsistent);
```

You can consult operations' ids on Operations subsection

ADA facade

The next table resumes ADA methods:

Method syntax	Comments
`loadDocument(String path): AbstractDocument`	Loads a document from the specified path
`writeDocument(AbstractDocument doc, String path): void`	Writes the document on the specified path
`createOperation(String id): Operation`	Creates the operation by its id
`analyze(Operation op): void`	Analyzes the operation
`transformTo(String path1, String type1, String, String type2): String`	Returns a literal string containing the model in syntax type2
`transformTo(String f1, String f2): void`	Transforms file f1 into type of file f2, and saves into it.

Transforming models

ADA is able to transform a document on a file format into another file format. Currently, two file formats are supported: original xml syntax and wsag4people syntax. ADA usage to transforms is:

```
//1. Instantiate ADA
ADA ada = new ADA();

//2. translates from wsag (existing file) to wsag4people (new file)
ada.transformTo("Template.wsag", "Template.wsag4people");
```

ADA OSGi

There are no usage differences between ADA OSGi based and standalone version, except recovering ADA as a service on OSGi. You can read how to do this task on Quick start/OSGi subsection.

ADA web service

ADA WSDL provides many of the operations available on standalone and OSGi versions. Every operation of the WSDL corresponds with an analysis operation. For each operation, a model (or two) should be passed as parameter, since these operations are atomic. Now, we present equivalences between WS operations and standalone operations.

ADA WS operation	ADA operation
`checkDocumentConsistency`	`ConsistencyOperation`
`explainInconsistencies`	`ExplainNoConsitencyOperation`
`getLudicrousTerms`	`LudicrousTermsOperation`
`getDeadTerms`	`DeadTermsOperation`

`explainLudicrousTerms`	`ExplainLudicrousTermsOperation`
`explainDeadTerms`	`ExplainDeadTermsOperation`
`isCompliant`	`ComplianceOperation`
`explainNonCompliance`	`ExplainNoComplianceOperation`

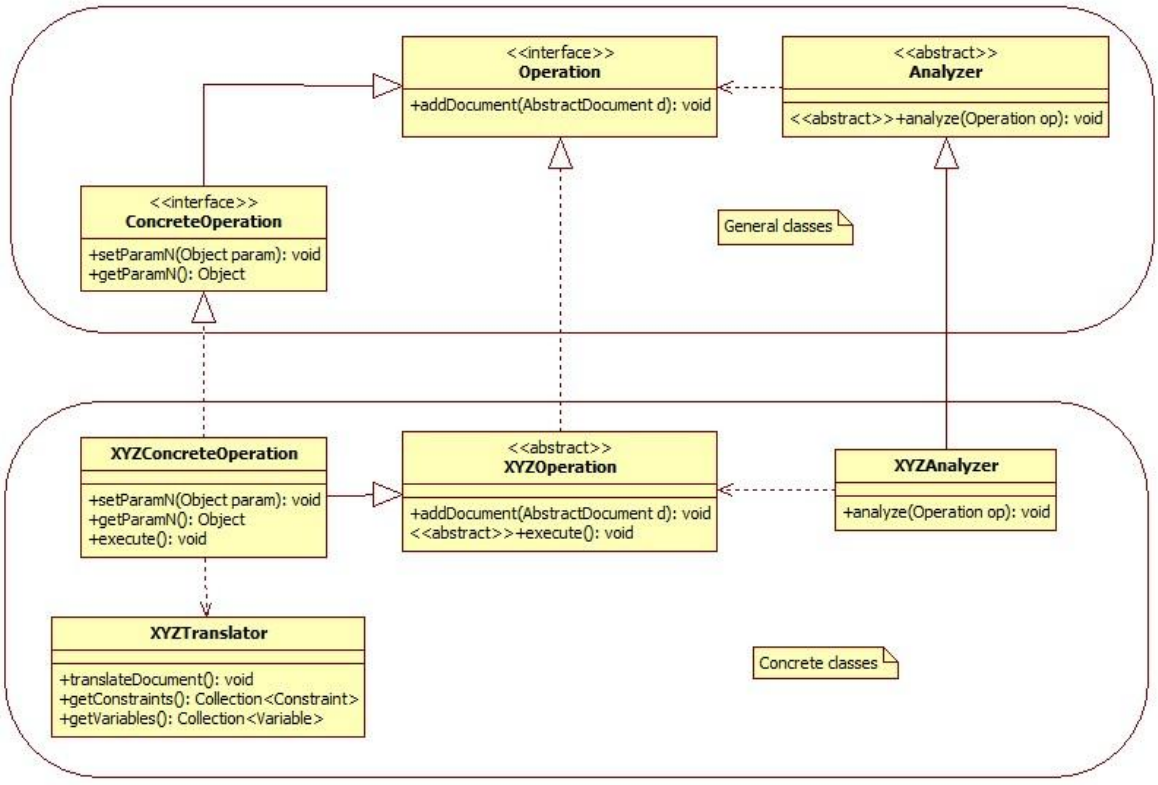
There are available also miscellaneous operations:

Miscellaneous operations	Description
`getMetricFile(String): byte[]`	Returns a metric file stored on the server
`addMetricFile(byte[]): String`	Uploads a new metric file to the server, and returns its path on it.
`xmlToWSAg4People(byte[]): byte[]`	Translates a document on xml format into wsag4people format
`wsag4PeopleToXML(byte[]): byte[]`	Translates a document on wsag4people format into xml format

Extending ADA

To extend ADA with a new analyzer, it is advisable to follow the indications of the next UML class diagram:

The three top elements are interfaces and abstract predefined classes of ADA's core, and the four bottom elements are concrete implementations for the new analyzer:



- XYZAnalyzer: extends abstract class Analyzer, and implements analyze method.
- XYZOperation: it is an abstract class. XYZOperation implements Operation and adds a new abstract method, execute, for all its sub instances
- XYZConcreteOperation: it implements a concrete operation and its methods, like compliance or consistency, and extends XYZOperation, overriding execute method
- XYZTranslator: an auxiliary class to translate the documents to a logical paradigm.

Troubleshooting

Front-end doesn't load!

Revise if you have installed java version 6. Check too if your browser is compatible with JavaFX (installation section).

Front-end becomes frozen when executing some operations!

Some operations (like compliance/explanations) are very computational expensive. Wait…

My document doesn't work at the front-end, or at the web service

Remember you have to use front-end metric file. Path to use is "*metrics/metric2575304161886105*". If ADA doesn't find your metric file, you loading will crash. Check also if your file has the correct syntax.

I can't upload metrics to the front-end

Although front-end button to upload metric files is enabled, we don't support currently it by security reasons. We are working on it.

I can't recover ADA instance from OSGi/ ADA throws an exception when using with OSGi.

Have you started ADAChocoAnalyser bundle? Remember to set start level 1 to ADACore and Choco plugins.

ADA standalone crashes!

You have to import also choco and antlr libraries. Don't forget to copy ADAConfig.xml to project root folder. And don't forget too locating metric files used on your documents!!! On the example presented in this guide, you have to copy metrics folder to project root folder.

Contact us

ADA team is composed by several persons. Now, we are two persons developing the tool: Antonio Jurado and Jesús García. On research level, Carlos Müller and Manuel Resinas are the "chiefs", under the supreme chief, Antonio Ruiz :-P.

You can visit tool website often www.isa.us.es/ada (tool web).

Further work

We are working to improve ADA. We are developing integrations with other tools too. Visit often www.isa.us.es/ADA to stay informed.

Acknowledgements

We want to thank everyone implied in ADA project, and all users that have given feedback about the tool ☺

Glossary

- Analyser: ADA component that implements one or more operations
- Creation constraints: a section on templates where you can define constraints over variables
- Dead term: A term is considered as dead when its qualifying condition is never hold since it collides with other elements of the document.
- Explanations: set of causes that make an error on a document/s. The error can be an inconsistency, a dead term, a ludicrous term or a non-compliant error.
- Guarantee Term: a term about a non-functional guarantee, which can include a condition for activation.
- Item: a type of constraint defined on creation constraints section of templates.
- Ludicrous term: a term is considered as ludicrous when its qualifying condition is hold but the service level objective is contradictory with other terms.
- Offer: response to a template that should match a template
- Operation: action to obtain information about a model
- Qualify Condition: a condition that has to be satisfied to take into account its SLO
- Service Description Term: service description terms can be considered as named, service–related sets of service features with their corresponding values
- Service Properties: Set of variables about functional and non-functional elements of the agreement.
- SLA (Service Level Agreement): a set of terms that state assertions about functional features and non-functional guarantees that service providers and service consumers must fulfill during the provision of the service
- SLO (Service Level Objective): an assertion defined over monitorable variables defined in the service properties section of the agreement document, and over external factors such as date, time, etc.
- Template: document where a provider or consumer shows its agreement propose
- Term compositor: a term composed by other terms
- Term: an element that includes information about non-functional guarantees, like response time or cost.
- WSAg 4 people: human-readable syntax of agreement documents
- WS-Agreement: a protocol specification for establishing agreement between two parties

References

- ADA website: www.isa.us.es/ADA
- Apache felix: http://felix.apache.org
- Choco solver: www.emn.fr/z-info/choco-solver/
- Equinox: www.eclipse.org/equinox
- ISA research groups: www.isa.us.es
- Java Runtime Environment: http://www.oracle.com/technetwork/java/javase/downloads/index.html
  WS-Agreement specification: http://www.ogf.org/documents/GFD.107.pdf