Nadia Dialogue Description (Dialogue Model)

The basis for Nadia is a dialogue model. A dialogue designer needs to specify a dialogue description that the Nadia System can process. There are two approaches to the creation of such descriptions:

- XML
- Java

In order to create the model in Java you should use the nadia-model project that you also find on Github. The definition in XML is a bit more effortful and error-prone. That's why we are planning to create a graphical development environment that helps you to create the XML file without having to write XML yourself.

When you specify the model in Java, you can use the `save()` or `saveAs()` methods to generate an XML representation that you can load Nadia with.

Dialogues, Tasks and ITOs

Every dialogue consists of tasks, i.e. a group of questions that ask information that are needed to execute this very task. A task can be to get the weather or to get trip information. Every XML file defines exactly one dialogue, that can consist of several tasks that consist of ITOs. An ITO is an information transfer object and represents the system question and the possible user answers to that question.

A very simple dialogue can be created like this:

```
    Dialog dialog = new Dialog("helloworld");

    //a dialog consists of tasks
    Task task1=new Task("getCities");
    dialog.addTask(task1);

    //a task consists of ITOs (Information Transfer Objects)
    ITO ito;
    AQD aqd;

    //1
    ito=new ITO("getDepartureCity", "Where do you want to start?", false);
    task1.addITO(ito);
    //an ITO is associated with AQDs (Abstract Question Description)
    aqd=new AQD();
    aqd.setType(new AQDType("fact.named_entity.non_animated.location.city"));
    ito.setAQD(aqd);

    //2
    ito=new ITO("getDestinationCity", "Where do you want to go?", false);
    task1.addITO(ito);
    //an ITO is associated with AQDs
    aqd=new AQD();
    aqd.setType(new AQDType("fact.named_entity.non_animated.location.city"));
    ito.setAQD(aqd);
```

This will generate a dialogue (helloworld) that consists of one task (getCities). This task consists of two questions: one that asks for the departure and one that asks for the destination city (assuming that we are going to build a travel information system). If you save this, the XML will look something like this (this extract is a bit simplified):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dialog name="helloworld">
<tasks>
    <task name="getCities">
        <itos>
```

```
            <ito name="getDepartureCity">
                <AQD>
                    <type>
                        <answerType>fact.named_entity.non_animated.location.city</answerType>
                    </type>
                </AQD>
                <fallback_question>Where do you want to start?</fallback_question>
                <useLG>false</useLG>
            </ito>
            <ito name="getDestinationCity">
                <AQD>
                    <type>
                        <answerType>fact.named_entity.non_animated.location.city</answerType>
                    </type>
                </AQD>
                <fallback_question>Where do you want to go?</fallback_question>
                <useLG>false</useLG>
            </ito>
        </itos>
    </task>
</tasks>
</dialog>
```

If you start Nadia and do not specify a dialogue file, the default dialogue will be loaded. So if you want to use your just created dialogue, you need to run Nadia with `-f path/to/file/name.xml` or call https://localhost:8080/nadia/index2.html (here you can upload your own dialogue). Afterwards (if you have initialised your dialogue) you can take a look at a box visualisation of your dialogue under https://localhost:8080/nadia/engine/dialog/d1/xml.

So what can we do now with this dialogue? Actually, not much more than answering two questions. We have not specified actions that should be executed. However, we also did not specify the answer. Wait, we did! We specified an answer type `fact.named_entity.non_animated.location.city`. This lets the system now that we ask for cities. We do not need to specify a parser. The system already comes with interpretation facilities. However, at the moment and for this demo you can only use the following cities:

- Edinburgh
- Glasgow
- Aberdeen
- Portree
- Uig
- Balloch
- Inverness
- Malaig

At this point we also did not make use of language generation and instead used the fallback question. We will come back to language generation at a later point.

Selectors

If you have more than one task, the system needs to decide which task to use. This decision can be made with the help of task selectors. At the moment this is a simple list of keywords (bag of words).

After you have created a task,

```
Task task1=new Task("getWeatherInformation");
```

you can add a bag of words:

```
bagOfWords = new ArrayList<String>(Arrays.asList("weather","forecast", "temperature"));
task1.setSelector(new BagOfWordsTaskSelector(bagOfWords));
```

XML:

```
     <selector>
        <bagOfWordsTaskSelector>
           <word>weather</word>
           <word>forecast</word>
           <word>temperature</word>
        </bagOfWordsTaskSelector>
```

```
    </selector>
```

Actions

A dialogue system needs to do something of course. We want to carry out a certain action. Often this is the execution of a parameterised query to get information like the weather in a specific city. To execute actions, we use action objects. At the moment we have three different types:

- Java Class Loader (JavaAction): load and execute a compiled java class
- Groovy (GroovyAction): write a Groovy Script
- HTTP (HTTPAction): send a POST or GET request, get an XML answer and extract the result via XPath

To get the weather from openweathermap.org you need to access the following URL: http://api.openweathermap.org/data/2.5/weather?q=London&mode=xml. We can realise that by using an HTTP Action:

```
HTTPAction httpaction=new HTTPAction("#result");
httpaction.setUrl("http://api.openweathermap.org/data/2.5/weather");
httpaction.setMethod("get");
httpaction.setParams("q=London&mode=xml&units=metric");
httpaction.setXpath("/current/temperature/@value");
task1.setAction(httpaction); //assuming task1 has been created before
```

Of course we do not want to always get the weather in London. That's why we use the name of the city that the user has mentioned before:

```
httpaction.setParams("q=%getWeatherCity&mode=xml&units=metric");
```

The name after the percentage sign refers to the ITO where the user has been asked for the city name, i.e. percentage signs introduce references to ITOs and sharps refer to the name of the result variable. For HTTPActions this is always `result`. Moreover, we don't want the system to just say the temperature. Instead we expect a full sentence. That's why we need to change the `utteranceTemplate` in the constructor:

```
HTTPAction httpaction=new HTTPAction("The temperature in %getWeatherCity is #result degrees.");
```

If we combine this with an ITO the resulting XML may look like this:

```
<task name="getWeatherInformation">
     <selector>
        <bagOfWordsTaskSelector>
           <word>weather</word>
           <word>forecast</word>
           <word>temperature</word>
        </bagOfWordsTaskSelector>
     </selector>
     <itos>
        <ito name="getWeatherCity">
           <AQD>
              <type>
                 <answerType>fact.named_entity.non_animated.location.city</answerType>
              </type>
           </AQD>
           <fallback_question>For which city do you want to know the weather?</fallback_question>
           <useLG>false</useLG>
        </ito>
     </itos>
     <action>
        <httpAction>
           <returnAnswer>true</returnAnswer>
           <utteranceTemplate>The temperature in %getWeatherCity is #result degrees.</utteranceTemplate>
           <method>get</method>
           <params>q=%getWeatherCity&amp;mode=xml&amp;units=metric</params>
           <url>http://api.openweathermap.org/data/2.5/weather</url>
           <xpath>/current/temperature/@value</xpath>
        </httpAction>
     </action>
  </task>
```

Last edited by Markus Berg, 4 months ago