Author: Anna Homolka
Contact: mail@anna-homolka.de

Purpose of the programm

This program is an extension for PyWPS which allows the chaining of WPS processes. The workflow is described as a XML document. This document can be sent to PyWPS, where it is translated into a PyWPS process and registered. Then it can be executed like a normal PyWPS process.

Structure of the XML document

The XML has three parts, the inputs section, the section where the processes are described and the outputs section.

Every workflow starts with the root element “workflow” and the attribute “identifier” which is the name of the workflow with which it can later be called as a PyWPS process.

```
<workflow xsi:noNamespaceSchemaLocation="process_chain.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" identifier="example">
```

In the inputs section the inputs for the process chain are defined. These inputs can be passed to the workflow when it is called as a PyWPS process.

```
<inputs>
    <input title="erster Input" type="literal" localIdentifier="Input1" defaultValue="5"/>
    <input title="zweiter Input" type="literal" localIdentifier="Input2" defaultValue="4"/>
    <input title="dritter Input" type="literal" localIdentifier="Input3" defaultValue="6"/>
</inputs>
```

It starts with the inputs tag. Within the inputs element the different inputs with its attributes are defined.

Every input tag has to contain the attribute “title”, which is the process title, the attribute “type” (possible values are literal and complex, corresponding to the possible WPS input and output types) and the attribute “localIdentifier”. This is the variable with which the input can be referenced within the process chain. Furthermore the attribute “defaultValue” can be defined.

The section where the processes are described contains all information to start the processes and defines the relationships between there inputs and outputs.

```
<try>
  - <startProcess identifier="dummyprocess" sleepSecs="1" synchron="False" processID="dummyprocess_1" service="http://localhost/cgi-bin/pywps.cgi">
        <input identifier="i1" sourceIdentifier="Input1"/>
        <input identifier="i2" sourceIdentifier="Input2"/>
        <input identifier="i3" sourceIdentifier="Input3"/>
        <output identifier="output1" localIdentifier="x"/>
    </startProcess>
  - <catch>
      - <startProcess identifier="dummyprocess" sleepSecs="1" synchron="False" processID="dummyprocess_1" service="http://localhost/cgi-bin/pywps.cgi">
            <input identifier="i1" sourceIdentifier="Input1"/>
            <input identifier="i2" sourceIdentifier="Input2"/>
            <input identifier="i3" sourceIdentifier="Input3"/>
            <output identifier="output1" localIdentifier="x"/>
        </startProcess>
    </catch>
</try>
```

Every “startProcess” element has to be inside a try/catch block. The process within the “try” tags is executed first and if it fails the process within the “catch” tags is executed. Like in the example they can be identical; it is just a formalism of the XML description which leads to this doubling.

The information needed to invoke the process is contained in the attributes of the “startProcess” tag. The possible attributes are listed below.

Author: Anna Homolka
Contact: mail@anna-homolka.de

Attribute	Description
identifier	Identifier of the process like in the getCapabilites description
processID	Name for the process within the process chain
service	URL of the WPS service
synchron	True for synchronous False for asynchronous process execution
sleepSecs	Count of sleep Seconds when the process is executed asynchronous

The attributes “identifier”, “processID” and “service” are mandatory, the attributes “synchron” and “sleepSecs” are optional. If a process contains just literal input, it will be executed asynchronous. If you want to execute it asynchronous “synchron” has to be set to “False”. If the process contains complexInput it will always be executed asynchronous. “sleepSecs” is the count of sleep seconds for an asynchronous process execution. The default is 3 s.

The inputs for the process are described in child elements of the “startProcess” element. There are three ways to describe a process input. All of them have in common, that they need the attribute “identifier” which is the name of the input specified in the process. The difference is the source of the input.

1. The process input is an input of the workflow.
   Then the attribute “sourceIdentifier” is needed. It gets the value of the attribute “localIdentifier” of the corresponding workflow input.

```
<inputs>
    <input title="erster Input" type="literal" localIdentifier="Input1" defaultValue="5"/>
    <input title="zweiter Input" type="literal" localIdentifier="Input2" defaultValue="4"/>
    <input title="dritter Input" type="literal" localIdentifier="Input3" defaultValue="6"/>
</inputs>
...
    <input identifier="i1" sourceIdentifier="Input1"/>
```

2. The process input is an output of a previous process.
   Then the attributes “sourceName” which is the identifier of the process output and “sourceProcess” which is the processID of the process the output is from.

```
<input identifier="i1" sourceName="output1" sourceProcess="dummyprocess_1"/>
```

3. The process input is an output of a previous process and got a reference.
   It is possible to give an output of a process a specific name with which it can be referenced in the workflow. This is explained in the next paragraph. This output can be used as an input by using the attribute “localIdentifier” with the name of the output.

```
<startProcess identifier="dummyprocess" sleepSecs="1" synchron="False" processID="dummyprocess" service="http://localhost/cgi-bin/pywps.cgi">
    <input identifier="i1" sourceIdentifier="Input1"/>
    <input identifier="i2" sourceIdentifier="Input2"/>
    <input identifier="i3" sourceIdentifier="Input3"/>
    <output identifier="output1" localIdentifier="x"/>
</startProcess>
...
<output identifier="output1" localIdentifier="x"/>
```

As mentioned the outputs of a process can be given a specific name with which they can be referenced in later processes. This looks like this:

```
<output identifier="output1" localIdentifier="x"/>
```

Author: Anna Homolka
Contact: mail@anna-homolka.de

The outputs of the process chain can be defined like the inputs for the process.

```
<outputs>
    <output type="literal" sourceIdentifier="output1" sourceProcess="dummyprocess"/>
    <output type="literal" sourceIdentifier="output" sourceProcess="dummyprocess2"/>
    <output type="literal" localIdentifier="x"/>
</outputs>
```

Either they are described with “processID” and the “identifier” of the process they are from or with the “localIdentifier”.

An example workflow “example.py” can be found in the tests folder of PyWPS.

Registration of process Chain

There are two ways to translate the XML into a PyWPS process. The first way is to call “manuellParser.py” which is located in the “processes” directory. The command is

python manuellParser.py “location of XML” “Name of process chain”

This command will translate the XML example.xml to the process chain example.py.

$ python manuellParser.py /home/user/shared/example.xml example.py

Don’t forget to write the name of the process (in this case example) to the “_init_.py” file. Then the process chain can be called like a normal PyWPS process.

There is also the possibility to send the XML as an input of the process “recieveXML” to the webserver.
In order to do this, the apache webserver needs writing access to the “processes” directory. This has to be done first.

The execute request can look like this:

```
localhost/cgi-bin/pywps.cgi?service=wps&version=1.0.0&request=Execute&identifier=recieveXML&datainputs=[password=1111;data=/home/user/shared/example.xml]
```

The inputs are the location of the XML and a password. If you want to change the process you can change the content of the XML and execute recieveXML again, with the same password. This mechanism should prevent other processes from being overwritten.