EUROPEAN COMMISSION

Community research

Climate Information Portal

CLIPC DELIVERABLE (D -N°: 4.1)

Toolbox interface specification – version 1.2

File name: CLIPC-WP4-D41-Toolbox-interface-specification.pdf}

Dissemination level: PU (public)

Author(s): *Wim Som de Cerff*
Maarten Plieger

Reviewer(s): *Martin Juckes*
Johannes Lueckenkoetter

Reporting period: *01/12/2013 – 31/05/2015*

Release date for review: ***1/10/2014***

Final date of issue: ***08/07/2015***

Revision table			
Version	**Date**	**Name**	**Comments**
0.1	19/5/2014	Som de Cerff	Initial document
0.2	11/9/2014	Som de Cerff	Updates throughout the document; Comments Maarten;
1.0	1/10/2014	Som de Cerff	Version provided for EC review
1.1	20/5/2015	Som de Cerff, Maarten Plieger	Updates throughout the document based on discussions with WP8 at Schiphol and Ispra workshops.
1.2	8/7/2015	Som de Cerff, Maarten Plieger	Updates based on comments from Martin and Johannes.

Abstract

This document defines the interface for the indicator toolbox calculation components. This interface specification will ease the integration of the indicator toolbox calculation components into the CLIPC architecture. To keep the maximum flexibility in implementing these components and reusing existing international standards it is proposed to use the OGC Web Processing Standard 1.0.0 for encapsulating the components for calculations and the OGC Web Coverage Service 1.0.0 for data extraction and re-gridding.

grant agreement n°607418

Table of contents

Executive Summary

Goal of the CLIPC toolbox interface specification is to describe the a set of requirements on the CLIPC indicator toolbox developed in WP8 to be able to ease integration, configuration and updates in the CLIPC portal infrastructure.

As the CLIPC portal infrastructure and the toolbox are developed in parallel, there will be updates to this document. The CLIPC portal infrastructure is described in D3.1 Conceptual design overall portal website with functional specifications. Also the specification documents from WP7 and WP8 have influenced the contents of this document.

To keep maximum flexibility in the CLIPC architecture existing international standards are used. These are the OGC Web Processing Standard 1.0.0 for encapsulating the calculation components, the OGC Web Coverage Standard for preparing input data and the OGC Web Mapping Service 1.3.0 for visualization.

For integration of the toolkit in the CLIPC architecture all of these services are relevant. For the actual components delivered by WP8 for performing calculations only the WPS services are relevant.

The document is intended for the toolbox calculation component developers to provide them the interface constraints.

1. Introduction

1.1 Purpose of document

This document defines the interface between CLIPC portal toolbox and the algorithms running in it: the toolbox calculation components.

1.2 Purpose of the interface description

According to the [DoW], Task 4.2 is responsible for the integration the toolbox calculation components developed by WP8 into the projects' infrastructure as INSPIRE compliant services so they can be used to generate new climate indicators.

These calculation components are developed to perform specific calculations based on indicator input data: comparing, ranking and aggregation of indicators. In order to integrate the toolbox calculation components, the interfaces need to be specified.

WP8 is responsible for quality control of the output product. Task 4.2 will integrate the toolbox calculation components in the web interface.

1.3 Structure of document

The toolbox interface specification document is setup according the ECSS-40 software engineering standard template for Software Interface Specification document [ECSS-40].

Chapter 1 describes the scope and intended readership for the document.
Chapter 2 describes the applicable and reference documents.
Chapter 3 describes the used abbreviations and definitions.
Chapter 4 describes the architecture the toolbox has to be fit into.
Chapter 5 describes the design and requirements for the toolbox interface

1.4 Intended readership

This document is intended for the developers of the indicator toolkit calculation components (WP8).

2. Applicable and reference documents

2.1 Applicable documents

[DoW]: Annex I - CLIPC Description of Work
[Arch] : WP3/4/5 Architecture Team report version 1
[D3.1] : D3.1 Conceptual design of the CLIPC portal
[D7.1] : D7.1 review of climate impact indicators
[D8.1] : D8.1 Impact models and aggregations
[M34] : MS34 Outline WP8 tools

All CLIPC milestones and deliverable documents can be found on:
http://www.clipc.eu/the-project/public-deliverables-and-milestones

2.2 Reference documents

[ECSS-40]: ECSS-40 Software Engineering standard
[WPS]: OGC-WPS 1.0.0, http://www.opengeospatial.org/standards/wps
[WMS]: OGC-WMS 1.3. 0, http://www.opengeospatial.org/standards/wms
[WCS]: OGC-WCS 1.0.0, http://www.opengeospatial.org/standards/wcs

3. Definitions and abbreviations

3.1 Definitions

Toolbox Calculation Component	A Toolbox Calculation Component is the smallest granule of execution code that is under control of the Toolbox architecture. It consists of an algorithm that needs to be executed on input data (one or more indicators) with provided settings as command line arguments and stores results in output data (one or more indicators). A specific Toolbox Calculation Component has no concern on how to obtain its input files, command line arguments and when to run. This is taken care of by the Toolbox architecture (see figure 1 and 2).

3.2 Abbreviations

Climate4impact	Web portal to access the ESGF and other climate model data, specifically targeted at the climate impact researcher: www.climate4impact.eu
ESGF	Earth Science Grid Federation, www.esgf.org
IS-ENES	Infrastructure for the European Network of Earth System Modelling, verc.enes.org
OGC	Open Geospatial Consortium
WMS	OGC Web Mapping Service standard
WPS	OGC Web Processing Service standard

4. Architecture

4.1 Introduction

The CLIPC Portal architecture is in development and is described in [Arch] and [D3.1]. In this document only the relevant parts for the toolkit will be described: the CLIPC toolkit integration architecture. The CLIPC will be a distributed architecture, where data services, processing services, view services and the web interface not necessarily are running at the same location.

As specified in [DoW] the main functions of the toolkit are

1. Tools for scenario-based exploration of climate change impact indicators;
   Goal of this part of the toolkit is to provide the user a set of flexible tools that allow users to explore impact data for e.g., different combinations of climate scenarios and socio-economic scenarios.

2. Tools for comparing, ranking and aggregating climate change impact indicators
   Goal of this part of the toolkit is to extend the exploration tool with the possibilities to compare, rank and aggregate climate change impact indicators to create personal cross cutting impact indicators, defined along predefined lines, e.g. combined impacts (a) of certain sets of climate stimuli, (b) of a combined set of extreme weather event based impacts, (c) on whole sectors like economy, society, natural environment and (d) on narrower sub-fields like particular population groups, habitat types, etc. Base input data will be the predefined indicators.

Each of these functional blocks need to be integrated into the CLIPC architecture, but only the second contains Toolkit Calculation Components.

In general, OGC services form the foundation of the CLIPC architecture:

- The OGC WPS interface specification provides the plug-in framework to be used for toolkit calculations.
- The OGC WCS interface specification provides the services to prepare the input data (subsetting, regridding).
- The OGC WMS standard provides the service for generic map visualizations.

The PyWPS framework will be used as implementation of the WPS standard. The ADAGUC framework is used as implementation of the WMS and WCS standard. The ADAGUC WMS is INSPIRE compliant.

4.2 CLIPC Toolkit Architecture

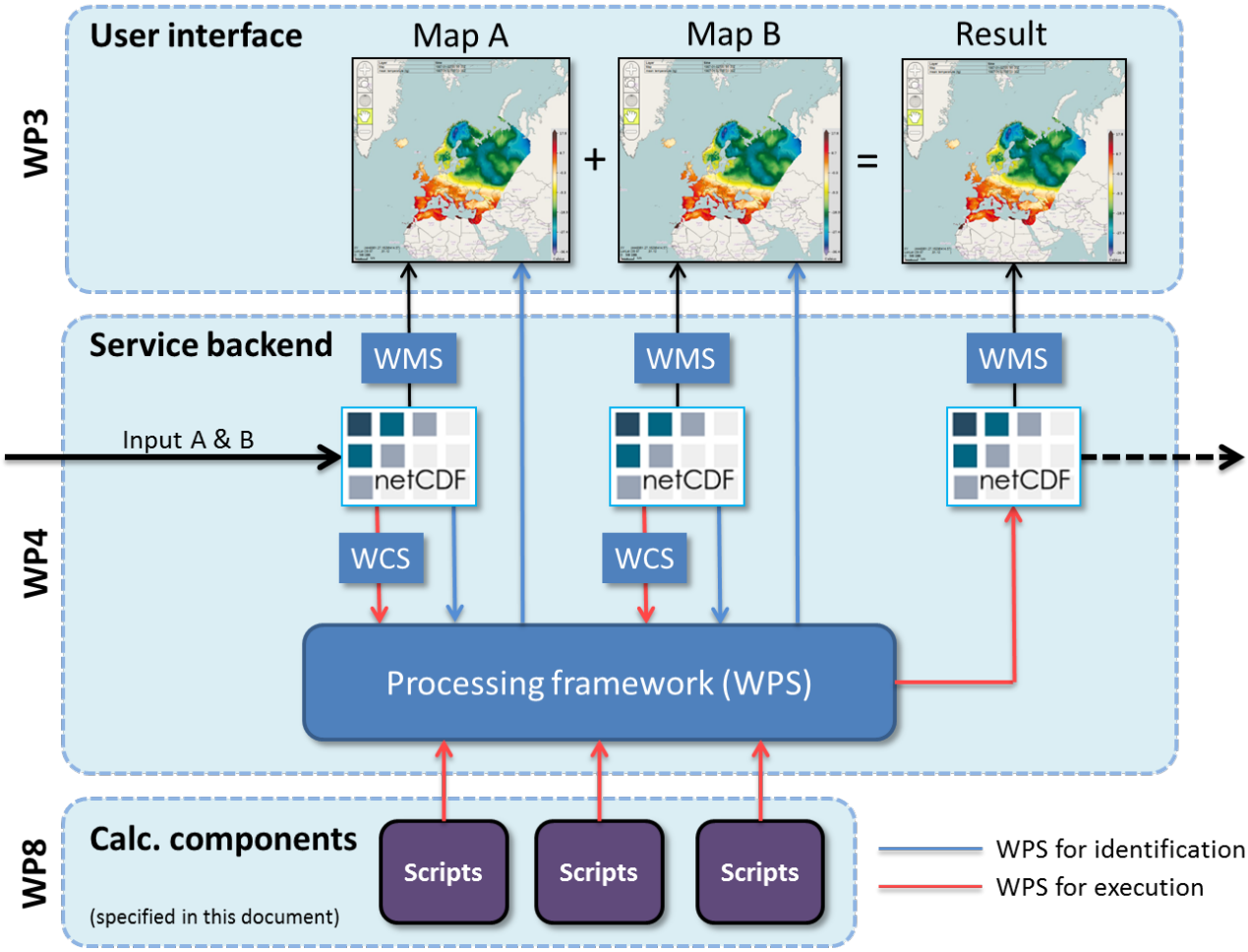

Figure 1 CLIPC toolkit architecture

In Figure 1 CLIPC toolkit architecture three different layers are shown:

1. The User Interface, here the user can select the indicators, select the operator for combining the indicators and different settings (scenario's, weights, etc.). It will also show the result to the user
2. The Service backend, this will collect the data, collect the settings, run the calculation component and collect the output data ready for visualization
3. The Calculation components, this are the algorithms for combining the indicators. It will use the input data, normalize it, perform the calculation and write the output data

The user interface is described in [D3.1]. In the next paragraph the service back end is described.

WPS Processing Framework :

Two types of WPS are used in the toolkit, one for identification and one for execution. For selecting parameters and adjusting settings in the toolkit, information about possible and valid

settings is required to create a responsive user interface. This information is obtained with the WPS for identification. It is used to collect information about input datasets and the calculation component. When the user adjusts settings in the toolkit, the execution WPS is started. The WPS for execution calculates the result, when finished the user interface displays the visualization of the result. All input and output data is stored in the netCDF format. Visualization is done using WMS and data extraction is done using WCS.

“WPS for identification” is used to retrieve information about the datasets. The identification WPS provides information about which parameters and settings are valid for the datasets, for example the time domain, geospatial coverage and available parameters. This information is required to be able to create a user-friendly interface. It helps to determine which settings are possible and what settings are valid ranges. The identification WPS also determines which settings are available and valid for the selected calculation component.

“WPS for execution” is used to start the calculation with given settings. This WPS gathers the input data using WCS and executes the selected calculation component. The result is a netCDF file, which can be visualized using WMS. This WPS executes the selected calculation component and monitors its progress.

5. Software and design

5.1 General provisions to the requirements in the IRD

This document is not based on a formal Interface Requirements Document (IRD) as specified in [ECSS-40]. It is decided not to write an IRD but to let the CLIPC Architecture team review the interface specification.

5.2 Interface requirements

Not applicable, see chapter 5.1.

5.3 Interface design

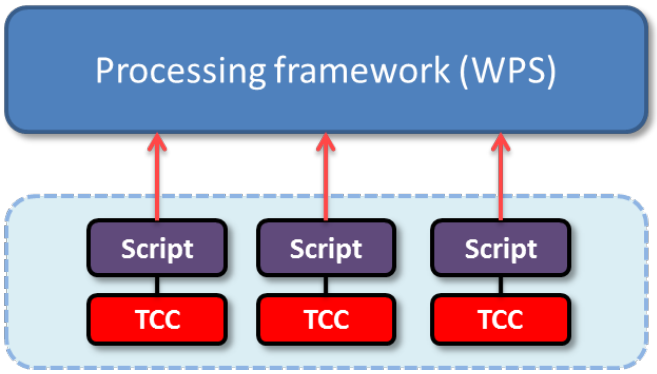

Figure 2 Toolkit Calculation Interface

A Toolbox Calculation Component is the smallest granule of execution code that is under control of the Toolbox architecture. It consists of an algorithm that needs to be executed on input data (one or more indicators) with provided settings as command line arguments and stores results in output data (one or more indicators). A specific Toolbox Calculation Component has no concern on how to obtain its input files, command line arguments and when to run. This is taken care of by the Toolbox architecture (see figure 2).

The Toolbox Calculation Component can be written in any programming language, as long as it provides a Linux command line executable program, which can be called by the WPS processing framework. The WPS wrapper scripts can be adjusted per TCC to adapt to specific logging and error codes, this enables to use existing toolbox calculation components without any adaptation. The WPS wrapper scripts have a very strict interface with the Processing framework, while the TCC's are loosely coupled. In this document guidelines on how TCC's should be constructed are given in chapters 5.7 and 5.8.

5.4 Standards used

Within the CLIPC Toolkit processing framework the following standards are used:

Purpose	Standard	Version	URL

Processing framework	OGC-WPS	1.0.0	http://www.opengeospatial.org/standards/wps
Visualization framework	OGC-WMS	1.3.0	http://www.opengeospatial.org/standards/wms
Data Extraction Framework	OGC-WCS	1.0.0	http://www.opengeospatial.org/standards/wcs

For implementing the framework, the following software packages are used:

Framework	Implements	URL
PyWPS	OGC-WPS 1.0.0	http://pywps.wald.intevation.org/
ADAGUC	OGC-WMS 1.3.0 OGC-WCS 1.0.0	http://adaguc.knmi.nl/

5.5 Additional interface requirements for WPS implementation

WPS 1.0.0 does not provide a guideline on how to cancel a running process. For CLIPC we extend here and add a method to stop a running process in a controlled (and secured) way. This will be implemented by using the process ID and an extra request call, which stops the process.

WPS jobs are executed as individual system processes by the operating system; they can safely be canceled by killing the process id without interfering other WPS jobs.

5.6 Security requirements

All services run behind in a secured environment and run behind access tokens. Access control is arranged by using OAUTH2.

5.7 Constraints on the Toolkit Calculation Components

The CLIPC can be a distributed architecture, where data services, processing services, view services and the web interface not necessarily are running at the same location. **Note that first CLIPC implementations will run in one location to ease developments.**

If a component runs at a remote location the following best practice constraints apply:

Nr	Constraint	Comment
1.	The component must run headless	No window manager interactions with the component needed to run the component.
2.	The component must provide progress and status information	Progress information should be given to indicate the user how far processing is completed. Progress information indicates a percentage and a short key message saying

		what step is currently being done. Update frequency should be around 1 second. This can be done as specified in Nr. 3.

3.	The component should provide progress and status information as JSON via stdout.	This information can be written regularly as consecutive lines to standard out in the following JSON format:
		`{ "progress":"<percentage complete>",`
		`"message": "<message with current task>"}`
		The WPS wrapper script can pass this information to the WPS framework. See the example in 5.9.

4.	The component must provide logging information.	A log file needs to be written separately. This log file is useful when something went wrong. The log file must indicate clearly which steps are completed and where the processing has failed for what reason.
5.	The component must return specified codes on success or failure: Exit code 0: success Exit code not 0: failure	Well-defined exit codes can be used to provide the end user information on results. If a failure occurs, it is possible to show the log file to the user.
6.	The component should return detailed codes on failure: Code 100: File(s) not found Code 101: Options incompatible Code 102: Variable not found in file Code 103: Time slice not found in file (this list will be maintained and updated)	Well-defined exit codes can be used to provide the end user information on results. If a failure occurs, it is possible to show the log file to the user.
7.	The component must write its outputs and log files in a specified working directory.	This will ease integration into the operational environment. Absolute paths should never be used. The TCC will write the log file and the WPS wrapper script knows how to read this file.
8.	The component must be able to run in parallel with other instances of the component.	As the component will be run through a web interface, multiple instances must be able to run in parallel (e.g., no .lock files in specific directories, standard temporary files in /tmp)

When the component needs to be integrated in the WP4 Processing environment, located at KNMI, additional constraints are applicable:

Nr	Constraint	Comment
1.	The component must run without runtime licenses.	No need for installing (commercial) licenses in the operational environment.
2.	The component must run in a Linux Redhat 6.x environment.	Currently the operational environment at KNMI
3.	The component must run using libraries, which can be installed using a standard package manager	No additional libraries need to be compiled.
4.	The component must be delivered with a test set.	To test the component off line in a testing environment before integration in the operational portal.
5.	The component must be delivered with documentation how to compile, install and test the component.	Essential to be able to integrate the component in the processing environment.

5.8 Interactions of toolkit calculation components with WPS framework

The WPS wrapper script controls the toolkit calculation component; these components interact directly with the input data. Data selection and finding required data in NetCDF files can be challenging, for example finding out which time indices to use with different calendars is difficult and error prone. This task can optionally be done by the WPS wrapper script which controls the TCC. In order to make time selection failsafe and correct, the wrapper script determines which time indices should be used. The time library in the ICCLIM package is able to deal with this problem and has been tested (and updated) with many datasets.

The following example shows how this information can be passed from the wrapper script to the TCC:

```
Tcc01.r -inputA <file name> -varA <variable name> -inputB <file name> -varB <variable name> \
                  -timeSliceA  <start:end:stride>  -timeSliceB <start:end:stride>  \
                  -option1 <value>
                  -ofile <file name>
```

In this example, the calling code specifies the variable name and the time indices that the TCC script should use, so that the script has an easy task to find the data it needs in each file. This makes it clear that the job of identifying which time indices to use (and checking compatibility of time axes) is with the wrapper script which calls the function.

The suggested mechanism is optional; e.g. wrapping existing tools (CDO,NCL,ADAGUC) in a WPS wrapper script is perfectly fine as long as constraints and functionality is maintained.

5.9 Example PyWPS wrapper script

For integration of the Toolkit Calculation Component, the following needs to be done:

- Define input and output parameters in the `__init__` function;
- In execute the Toolkit Calculation Component needs to be called; This can be an 'R' script, where `stdout` can be monitored by the Python script to get progress information.
- The R script can write results to a file

The example wrapper script process calculates the number '42' in 10 seconds[1], by calling an R program doing the calculation.

Goal is to show how an asynchronous PyWPS process can be made, which allows for progress monitoring. Below three scripts are listed:

- ultimatequestionprocess.r – The R script which thinks for 10 seconds and then writes the result to a file in the local directory
- ultimatequestionprocess.py – Starts and monitors the R script, implements pywps.WPSProcess , reads the file which was created by R
- ultimatequestionprocess_test.py – Optional script to test above processing using commandline

```
##### ultimatequestionprocess.r #####
for (counter in 1:9){
  Sys.sleep(1)
  print (paste("{'progress':",
               "'",
               counter*10,
               "',",
               "'message':",
               "'Thinking...'}",
               sep=""))
}
write("42",file="out.dat")

# ------------------------------------------------------------------------------------------

##### ultimatequestion.py #####
from pywps.Process import WPSProcess
import sys
from subprocess import PIPE, Popen, STDOUT
from threading  import Thread
import time
import json
import os

""" Helper function to execute a commandline tool (E.g. Rscript) and monitor its output
asynchronousely """
def executeSystemProcess(cmds,callback=None,env = None,bufsize=0):
  try:
      from Queue import Queue, Empty
  except ImportError:
      from queue import Queue, Empty  # python 3.x
```

[1] This script was created by Jorge de Jesus (jorge.de-jesus@jrc.it) as suggested by Kor de Jong.

```
  ON_POSIX = 'posix' in sys.builtin_module_names

  def enqueue_output(out, queue):
      for line in iter(out.readline, b''):
          queue.put(line)
      out.close()


  p = Popen(cmds, stdout=PIPE, stderr=STDOUT,bufsize=bufsize, close_fds=ON_POSIX,env=env)
  q = Queue()
  t = Thread(target=enqueue_output, args=(p.stdout, q))
  t.daemon = True # thread dies with the program
  t.start()

  # read line without blocking
  while True:
    try:  line = q.get_nowait()
    except Empty:
        if(t.isAlive() == False):
          break;
    else: # got line
      if(callback != None):
        callback(line)
    time.sleep(100./1000000.0)
  return p.wait()

""" The class Process which implements WPSProcess from PyWPS
    It calls the R script "ultimatequestionprocess.r" and monitors it output asynchronousely.
    The status and progress output of the rscript is a json line written to stdout containing
a 'progress' and 'message' object, like {'status':'thinking','progress':'45'}
    The final result is written by the rscript to a file called out.dat, this file is being
read by this process to find the answer
"""
class Process(WPSProcess):
    def __init__(self):
         # init process
        WPSProcess.__init__(self,
                            identifier="ultimatequestionprocess", #the same as the file name
                            title="Answer to Life, the Universe and Everything Calculated with
R",
                            version = "2.0",
                            storeSupported = "true",
                            statusSupported = "true",
                            abstract="Numerical solution that is the answer to Life, Universe
and Everything. The process is an improvement to Deep Tought computer (therefore version 2.0)
since it no longer takes 7.5 milion years, but only a few seconds to give a response, with an
update of status every 10 seconds.",
                            grassLocation =False)

        self.Answer=self.addLiteralOutput(identifier = "answer",
                                          title = "The numerical answer to Life, Universe
and Everything")



    def execute(self):
        # Callback function, to be passed to the executeSystemProcess routine
        def callback(message):
          try:
            message = message[5:-2].replace("'","\"");
            data = json.loads(message)
            self.status.set(data["message"],data["progress"]);
          except:
            self.status.set("Cannot decode json: "+message,0);
            pass

        self.status.set("Preparing....", 0)

        cmds = ["Rscript",os.environ['PYWPS_PROCESSES'] +"/ultimatequestionprocess.r"]

        executeSystemProcess(cmds,callback);

        f = open('out.dat', 'r')
```

```
        processingResult = f.read( )
        f.close()

        #The final answer
        self.Answer.setValue(processingResult)


# --------------------------------------------------------------------------------------------

##### ultimatequestionprocess_test.py #####
""" Make sure these environment variables are set before executing:
export PYWPS_CFG=wps/pywps.cfg;
export PYWPS_PROCESSES=wps/processes;
export PYWPS_TEMPLATES=wps/pywps-3.2.1/pywps/Templates
export PYTHONPATH=wps/pywps-3.2.1/build/lib
"""

from pywps.Process import WPSProcess
from pywps.Process import Status
import ultimatequestionprocess as ProcessToTest

#Override status class and method in order to print to stdout directly.
class MyStatus(Status):
  def set(self,string,p):
    print(string+" percentage complete:"+str(p))
status = MyStatus();

p=ProcessToTest.Process()
p.status=status
p.execute()

print "The answer of the ultimatequestionprocess is: "+p.Answer.value
# --------------------------------------------------------------------------------------------
```