EIAO
European Internet Accessibility Observatory
Project no.: IST-2003-004526- STREP

EIAO documented source code

Deliverable Number: D5.2.2.1-2

Version:	1.8
Date:	2008-05-08
Author:	Nils Ulltveit-Moe, Morten Goodwin Olsen, Anand B. Pillai, Terje Gjøsæter
Dissemination Level:	PU
Status:	FINAL

This document consists of 112 pages including this cover (2 pages).

Version Control

Version	*Status*	*Date*	*Change*	*Author*
0.1	DRAFT	2007-05-31	First draft	Nils Ulltveit-Moe
0.2	DRAFT	2007-06-07	Added Anands description of upgrade to new HarvestMan version.	Nils Ulltveit-Moe Morten Goodwin Olsen
0.3	DRAFT	2007-08-16	Added section on fallback handling for documents with wrong content type.	Anand B. Pillai
0.4	DRAFT	2008-02-19	Restructuring according to latest design.	Nils Ulltveit-Moe
0.5	DRAFT	2008-02-20	Merged in comments from Morten, Annika and Terje. Restructured first part of the document including guided tour.	Morten Goodwin Olsen
0.6	DRAFT	2008-02-21	Restructuring of the middle part.	Nils Ulltveit-Moe
0.7	DRAFT	2008-02-21	Restructuring finished.	Nils Ulltveit-Moe, Morten Goodwin Olsen
0.8	DRAFT	2008-02-22	Updated after inspection.	Nils Ulltveit-Moe, Terje Gjøsæter, Annika Nietzio, Morten Goodwin Olsen
0.9	DRAFT	2008-02-26	Added and updated detailed implementation information.	Nils Ulltveit-Moe
1.0	DRAFT	2008-02-27	Added detailed implementation information (WAMs and other modules)	Nils Ulltveit-Moe, Morten Goodwin Olsen
1.1	RC	2008-02-29	Added module descriptions.	Nils Ulltveit-Moe Morten Goodwin Olsen
1.2	RC	2008-03-10	Added description of utility modules, data and on-line reports.	Nils Ulltveit-Moe, Morten Goodwin Olsen, Terje Gjøsæter
1.3	RC	2008-04-01	Documented utility functions and improved Relaxed documentation. Final review of entire document.	Nils Ulltveit-Moe, Morten Goodwin Olsen
1.4	FINAL	2008-04-04	Final comments	Morten Goodwin Olsen, Nils Ulltveit-Moe
1.5	FINAL	2008-04-16	Added description of CSS caching functionality.	Anand B. Pillai, Nils Ulltveit-Moe
1.6	FINAL	2008-04-24	Updated default scope rule.	Nils Ulltveit-Moe

				Morten Goodwin Olsen
1.7	FINAL	2008-05-08	Updated GUI installation.	Terje Gjøsæter, Nils Ulltveit-Moe
1.8	FINAL	2008-05-08	Final comments	Mikael Snaprud

1. Table of Contents

1 Introduction..... 6
1.1 Brief project description..... 6
1.2 Scope of this document 6
1.3 Related work and readers instructions..... 6
2 Installing, configuring and running the Observatory..... 7
2.1 Installation guide..... 7
2.2 System Configuration..... 9
2.3 Installing the WAM separately..... 11
2.4 Upgrade of the Observatory..... ...12
2.5 Installation of the On-line Reporting Tool (GUI)..... 13
2.6 Managing crawls with the eiaoassess client program..... 16
3 Observatory Overview..... 18
3.1 Hardware specifications..... 19
3.2 Guided tour..... 19
3.3 Pre-caching of on-line reports..... 28
4 EIAO architecture..... 30
4.1 Site URL server..... 30
4.2 URL repository..... 30
4.3 Crawler manager..... 31
4.4 Crawler..... 31
4.5 Sampling server..... 31
4.5.1 Sampler..... 31
4.6 WAM server..... 32
4.7 ETL server..... 32
4.7.1 ETL..... 33
4.8 Data Warehouse..... 33
4.9 On-line reports..... 33
5 Details on Observatory parts..... 34
5.1 SiteURLServer - Site URL server..... 34
5.1.1 Starting a test run..... 36
5.1.2 Aborting a test run..... 37
5.1.3 Persistent state..... 38
5.1.4 Eiaoassess command..... 38
5.1.5 Site URL server modules..... 38
5.2 URL_repository - URL repository..... 39
5.2.1 Scoping rules (include / exclude patterns)..... 41
5.2.2 Example of scope rules in the URL repository..... 41
5.2.3 URL repository modules..... 42
5.3 Crawlers - Crawler manager..... 43
5.3.1 Watchdog functionality for hanging crawlers..... 44
5.3.2 Watchdog functionality for crawlers that exceed memory usage..... 46
5.3.3 Crawlers module..... 48
5.4 Crawler – EIAO branch of HarvestMan crawler..... 48
5.4.1 Crawler and fetcher thread state machines..... 48
5.4.2 Last-modified timestamp handling..... 51
5.4.3 Least recently used cache..... 51
5.4.4 Crawling according to scope..... 51

5.4.5 URL collections to support framesets.....52
5.4.6 Crawler functionality for writing URLs to memcached and URL repository.....52
5.4.7 Functionality for checking if an URL already has been downloaded.....53
5.4.8 JavaScript handling.....54
5.4.9 Crawler modules.....54
5.5 SamplingServer - Sampling server.....56
5.5.1 Sampling server modules.....56
5.5.2 SamplingAlgorithm - Sampler.....56
5.5.3 SamplingAlgorithm modules.....57
5.5.4 AdaptiveSampling - CWAM, standard deviation and error margin calculations. .58
5.5.5 PyTripleStore - Python based triplestore.....58
5.5.6 PyTripleStore modules.....62
5.6 WAMs - WAM server.....63
5.6.1 WAM call interface.....64
5.6.2 WAMs/wamlib - WAM library.....65
5.6.3 Default return value for WAMs.....66
5.6.4 WAM library modules.....67
5.6.5 WAMs/relaxed_wam - EIAO Relaxed based WAM.....68
5.6.6 EIAO Relaxed WAM modules.....69
5.6.7 WAMs/CSSWam - CSS A-WAM.....70
5.6.8 CSS AWAM modules.....72
5.6.9 RelaxedWAM - Relaxed HTML validator based AWAM.....72
5.6.10 EIAO specific files or modified files in Relaxed.....73
5.7 ETLServer - ETL server.....75
5.7.1 ETL server modules.....76
5.8 Datawarehouse - Data Warehouse.....76
5.8.1 Datawarehouse/dw20load – ETL.....76
5.9 UserInterface - On-line reports.....77
5.9.1 Online Reporting Tool – view and controller.....77
5.9.2 DWReader2 - Data Warehouse GUI Model.....79
5.9.3 DWReader2 modules.....80
5.10 Other modules.....80
5.10.1 INSTALL - Installation module.....80
5.10.2 SystemConfiguration - System configuration module.....80
5.10.3 Monitoring - Scripts for monitoring ongoing test runs.....81
5.10.4 DBCleaner - Cleaning of logs and databases.....82
5.10.5 CronWAM - Cron job for WAM service monitoring.....84
5.11 Utility functions.....84
5.11.1 GDB utilities.....84
5.11.2 MemCache – 3rd party memcached client.....85
5.11.3 QA – Quality Assurance modules.....85
5.11.4 profiling – Source code profiling.....85
5.11.5 eiaotime – Time module patch to avoid internal OS error 514.....86
5.11.6 timeoutsocket - 3rd party patch to add timeout support for sockets.....86
5.11.7 redland/unicodepatch – Redland for Python unicode patch.....86
5.12 Other data.....86
5.12.1 EIAO RDF schema.....86
5.12.2 Input URLs.....87
5.12.3 Documentation.....87
6 Future improvements.....87
6.1 PDF WAM.....87

6.2 Animated GIF AWAM..... 87
6.3 Crawler web page caching..... 88
7 Appendix A – Scope Pattern List Schema..... 90
8 Appendix B – Instance of scope pattern list..... 91
9 Appendix C - Developer hints and tips 92
9.1 Command line options..... 92
9.2 Logs 93
9.3 Debugging the Observatory..... 95
9.3.1 Starting ETL server in no-write (dummy) mode..... 95
9.4 Remote Cpython debugger using Winpdb..... 95
9.5 Cpython debugging using GDB..... 96
9.6 Design rules..... 96
9.6.1 Design rule: Always use unicode internally..... 96
9.6.2 Design rule: Block instead of using active waiting..... 96
9.6.3 Design rule: Parametrise all constant values..... ...97
10 Appendix D Content statistics WAM (MWAM)..... 98
10.1 Updated RDF ontology..... 98
10.2 eiao:MetaData RDF class..... 98
10.2.1 eiao:MetaDataType RDF class..... 99
10.2.2 eiao:value property..... 100
10.2.3 eiao:rangeLocation class..... 100
10.2.4 RDF report showing the eiao:MetaData extension..... 100
10.3 Technology usage statistics..... 101
10.3.1 Technologies within a web page..... 102
10.3.2 Technologies a web page links to..... 102
10.4 Content filters (e.g. HTML Tidy) applied..... 103
10.5 Language..... 103
10.6 CSS media types..... 104
10.7 How to infer MIME types..... 105
10.7.1 How to infer MIME types from file extension..... 105
10.7.2 Inferring MIME type for applet element..... 105
10.8 Measurement data handling..... ..105
10.8.1 eiao:siteSurvey measurement data..... 106
10.8.2 eiao:scenario meta data (web page level)..... 107
10.8.3 Content statistics data for earl:TestSubject..... 107
10.8.4 earl:result measurement data..... 108
11 Appendix E Terms and acronyms..... .108

1 Introduction

European Internet Accessibility Observatory (EIAO) v 2.2 is a tool for performing regular large scale accessibility surveys on a set of web sites according to the UWEM 1.2 fully automatable rule set. The tool consists of a web crawler that crawls the web site using a breadth-first strategy, a Sampler that extracts a uniform random sample of the web pages crawled, Web Accessibility Metrics (WAM) for evaluation of web pages, an ETL that loads evaluation results from the evaluation RDF databases to the Data Warehouse and a web based on-line reporting tool. This document presents how to install and run the Observatory, its architecture and also how software components work in more detail, with reference to the source code modules involved.

1.1 Brief project description

Access for all to the Information Society is a declared key-goal for the European Union. With the wide-spread adoption of the Internet – in particular for access to government information and services, it is essential to secure access for all citizens. However, there are still digital barriers that make access to information difficult, especially for people with disabilities.
Large scale web accessibility benchmarking can help to locate potential barriers and fuel the development towards the Internet accessibility for all.

The EIAO (European Internet Accessibility Observatory) [EIAO] project, funded by the European Commission, has established the technical foundations of an Internet Observatory for large-scale evaluation of the accessibility of European websites.
The demonstrator based on the fully automatable test set defined in the Unified Web Evaluation Methodology, UWEM 1.2 [UWEM] which is based on WCAG1.0 [WCAG1.0].

1.2 Scope of this document

This document describes how to install and operate the EIAO Observatory. It also describes the SW architecture and describes the individual modules that the Observatory consists of.

1.3 Related work and readers instructions

UWEM 1.2
D3.3.2 WAM specification

Chapter 2 describes installation, configuring and running of the Observatory. Chapter 3 shows how the Observatory works through a typical use case of crawling, evaluating and loading one web site. Chapter 4 describes briefly the Observatory architecture and the different Observatory services and components. Chapter 5 describes each Observatory component and EIAO specific functionality in detail. An updated MWAM specification is included in Appendix D.

A user wanting to run crawls to perform large scale evaluations needs to read chapter 1 to 3. The rest of the document is information that is useful for developers wanting to modify or extend the Observatory.

2 Installing, configuring and running the Observatory

2.1 Installation guide

The Observatory has only been fully tested on Fedora core 6 and Fedora core 7[1].

Following is the installation description for Fedora core 7. A maintained version is available at http://svn.eiao.net/robacc/INSTALL

In order to install the Observatory on Fedora core 7, run the following commands as root.

(1) Prerequisite: Database setup

Make sure that mysqld and postgresqld are running. If they are not started at boot time, enable them with:

```
/sbin/chkconfig --level 3  mysqld on
/sbin/chkconfig --level 3  postgresql on
```

(On the standard FC7 installation, the daemons are not started at boot time.)

Update /var/lib/pgsql/data/pg_hba.conf to trust connections from localhost (and any other computer that should be able to access your databases). Typically pg_hba.conf should include:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
local   all         all                               trust
# IPv4 local connections:
host    all         all         127.0.0.1/32          trust
```

Note that this allows any user on your system to connect to all your databases! If this is not an appropriate security policy, refer to the PostgreSQL manual for other, more restrictive and secure, possibilities.

Update /var/log/pgsql/data/postgresql.conf to support 5000 connections:

```
max_connections = 5000
```

Update also the number of resources PostgreSQL can allocate. The exact numbers depend on your system, but the default configuration is too low.

```
shared_buffers = 512MB
work_mem = 64MB
maintenance_work_mem = 512MB
checkpoint_segments = 32
```

1 Additionally, both the crawlers and WAMs have been running successfully on Windows. However, we assume that to make the complete Observatory run on Windows, further modifications are required.

For PostgreSQL to work with the required number of resources, the shmmax and sem parameters of the kernel need to be updated.

Typically, /etc/sysctl.conf should include:

```
kernel.shmmax=549199872
kernel.sem=250     32000   32      512
```

For the new parameters to take effect run:

```
/sbin/sysctl -p
```

restart postgres
/etc/init.d/postresql restart
Check that postgresql is running OK. This can be done by the following:

```
tail -50 /var/lib/pgsql/pgstartup.log
```

No error messages should be present in this log.

MySQL configuration:

Update /etc/my.conf to allow larger packet sizes.
Add the following under
[mysqld]
...

```
set-variable=record_buffer=64M
set-variable=max_allowed_packet=64M
```

restart mysql

```
/etc/init.d/mysqld restart
```

(2) Prerequisite: JDK from Sun: The Observatory should use Sun JDK 1.6, since JDK 1.5 and older versions has a bug that causes memory leak during exception handling in Java.

Download jdk-6u4-linux-i586-rpm.bin from http://java.sun.com/javase/6/ and install it.

(3) Installation of additional software packages

```
sh FC7Install.sh
```

This script will install additional packages, setup the SQL-databases, and download the EIAO sources.

(4) Installation and initial setup of EIAO software:

```
mkdir /etc/eiao
cp robacc/SystemConfiguration/initial.rdf /etc/eiao
```

Manually change the passwords in `initial.rdf`. Section 4.2 describes in more detail all different settings of initial.rdf.
These passwords include urlreppassword and dbpassword. 'removed' is not the intended password. Install the Observatory with:

```
cd /data/svn/robacc/
python setup.py install
```

During the initial installation, you will be asked to choose passwords for postgres. These should be the same as in `initial.rdf`.

2.2 System Configuration

The system configuration file `/etc/eiao/initial.rdf` contains settings used by the Observatory and needs to be adjusted accordingly. The settings are described in the table below:

Parameter	Example	Description
dwuser	eiaodw	Username for the Data Warehouse.
dwpassword	removed	Password for the Data Warehouse.
maxmemory	350	Max amount of information each Crawler is allowed to use in Mb.
dwdatabase	eiaodwr20	Data Warehouse Database
webproxy		Web proxy URL (optional)
webproxyuser		Web proxy username
webproxypassword		Web proxy password
urlrepusername	eiaourlrep	URL repository user name
urlreppassword	removed	URL repository password
urlrepdatabase	eiaourlrep	URL repository database
urlrephost	localhost	URL repository host
maxnotimeout	300	Watchdog keepalive timeout
samplingpolicy	fixed	Sampling policy: fixed for fixed number of samples, errrormargin to sample until a given error margin. (errormargin is deprecated due to statistical issues.)
etlserver	http://localhost:8890/	ETL server host
dbcleaner	http://localhost:8892/	DB cleaner host (removing unused logs)
samplingserver	http://localhost:8891/	Sampling server host

Parameter	Example	Description
rdfprefix	http://www.eiao.net/rdf/2.0/	RDF prefix, used by WAM and RDF-reader
memcache	localhost:11211	Memcached server
nuts3file	/etc/eiao/nuts3.csv	File with list of all NUTS categories
barriercomputationfile	/etc/eiao/barriercomputation_fcui.data	File with list of barrier computations (deprecated, may be removed in future versions.)
siteurlserver	http://localhost:8889	Site URL server host
pagetimeout	120	Maximum duration of a HTTP request
minpagessampled	30	Minumum number of pages sampled from a site for the sample to be accepted and loaded into the data warehouse. Note that if a web site has been verified to contain less than 30 pages, then the field smallsite in the page table of the URL repository must be set to True. It is by default False.
smtpserver	localhost	SMTP server
administratoremail	maintainers@localhost	Administrator e-mail
webcachedirectory	/var/local/cache/eiao/harvestman/storedfiles/	Directory for web cache
loglocation	/var/log/eiao/	Log files location
numberofsamples	600	Number of samples to be evaluated
maxurlsfromsite	6 000	Number of pages to be downloaded from the Crawler
homedir	/data/svn/robacc	Home directory of the Observatory
javadir	/usr/share/java-1.6.0	Location of Java. Note that version 1.6.0 or newer of Sun's jdk should be used.
jythondir	/usr/lib/Jython-2.1	Location of Jython. Note that the WAM requires Jython 2.1. It does not work with Jython 2.2 due to some problems/incompatibilities in the XML parsing libraries in Jython 2.2 that breaks SOAPpy.
wamservers	http://localhost:8888	WAM server host

Parameter	Example	Description
cssmediatype	screen	CSS media type used by the Observatory.
crawlerconfigtemplate	/etc/eiao/config.template	Configuration file for HarvestMan
configdirectory	/var/local/cache/eiao/harvestman/config/	Temporary storage for configruation files for HarvestMan

Table 1: System Configuration settings

2.3 Installing the WAM separately

It is possible to install only the WAM evaluation framework without installing the rest of the Observatory services. To install the WAM, do the following:

1) Check out[2] the Observatory from SVN:

```
mkdir /data/svn
cd /data/svn
svn checkout http://www.eiao.net/robacc
```

2) Set up the WAM configuration in /etc/eiao/initial.rdf. The following attributes must at least be set:
Note that the WAM configuration is also part of the System Configuration in chapter 2.2.

Parameter	Example	Description
homedir	/data/svn/robacc	Home directory of the Observatory
javadir	/usr/lib/jvm/java-6-sun	Java location
jythondir	/usr/share/jython	Jython location
wamservers	http://localhost:8888	Address the WAM listens to
cssmediatype	screen	CSS media type to be evaluated by the WAM.

Table 2: System Configuration Example for WAM

2 Note that a checkout requires user name and password of the SVN repository. If this is not present, svn export can be used. Further details can be found in [SVN].

3) Install the WAM:

```
cd robacc
python setup.py wam
```

4) Start the WAM (or wait 10 minutes for it to be started automatically by the cron job.)

```
/etc/init.d/relaxedwam start
```

5) Run the WAM conformance tests:

```
cd robacc/Test_suites/WAMConformance/
python test.py
```

All tests should pass.

2.4 Upgrade of the Observatory

If the EIAO software is already installed and you wish to upgrade it, go to the robacc directory:

```
cd /data/svn/robacc
```

And run SVN update:

```
svn update
```

Then, to deploy the packages, run:

```
python setup.py install
```

(Note that if only the WAM is installed, then you need to run `python setup.py wam` instead.)

Running Observatory services should be restarted manually after the upgrade.

The Observatory services are started automatically at server startup. Furthermore, the WAM service is automatically restarted when it is not responding – checked by cron jobs. The other services do not directly support automatic restarts by monitoring jobs, due to the fact that e.g. the Site URL server needs to be populated for it to work correctly.

2.5 Installation of the On-line Reporting Tool (GUI)

Install plone 2.5.5 with zope 2.9.* and five:

http://plone.org/products/plone/releases/2.5.5

For linux: use the unified installer with suitable versions of zope and five included:

https://launchpad.net/plone/2.5/2.5.5/+download/Plone-2.5.5-UnifiedInstaller.tgz

The above is tested and found to work with the GUI.

Create a zope-instance in `/opt/zopeinstance` using `mkzopeinstance.py`

If using the unified installer and there are no products in the Products folder, copy all products from the zeo-cluster that comes with the standard install.

Checkout the AccessibilityObservatory product from svn into the Products folder

```
cd /opt/zopeinstance/Products
svn co http://svn.eiao.net/robacc/UserInterface/GUI_Light/AccessibilityObservatory/
```

Make sure that all files are owned by the plone user, and that the plone user (user `plone` is assumed here, it is created by the unified installer) is also set as `effective-user plone` in the `zope.conf` file

```
chown -R plone.plone /opt/zopeinstance
```

Make sure that site-packages for the Python *used by the zope instance (the unified installer supplies its own python!)* contain all dependencies, if not, error messages will be obscure and not necessarily give any indication about what is wrong.

`DWReader2.py` and `DWError.py` from `http://svn.eiao.net/robacc/DWReader2/`

http://svn.eiao.net/robacc/WAMs/relaxed_wam/sc2.py

Edit `/etc/eiao/initial.rdf` to contain the correct values for `dwhost, dwdatabase, dwuser` and `dwpassword`

In addition, the following packages must be installed:

`psycopg` from http://www.initd.org/pub/software/psycopg/psycopg-1.1.21.tar.gz
The `mx` module containing `mxDateTime`: http://www.t2-project.org/packages/py-mx.html

Make sure the plone user have rights to write cache files to `/tmp`
Start the zope instance and enter the zope management interface (zmi)

Note that the paths below refer to the zmi folder structure and not the file system!

check that /Control_Panel/Products contains the AccessibilityObservatory product
From the root folder of zmi, create a new Plone site, choosing id and title = "guilight", base profile = "Plone site" and Extension profiles = AccessibilityObservatory
Check in /guilight/portal_setup, on the properties tab, that Active site configuration= AccessibilityObservatory, not Plone site
From /guilight create a new AccessibilityObservatory Content, type=ReportSection Id=reports

In the ZMI for the plone site select security. Under “Add Portal Member” **only** the following permissions should be selected: Manager. This ensures that only the manager can add users to the site. Note that this also means that acquire should not be selected.

Try to access `http://localhost:8080/guilight/reports` to check that the site is available.

For debugging: switch between real data and fake data by editing the configuration file of the product (not from zmi, from the filesystem):

```
vi /opt/zopeinstance/Products/AccessibilityObservatory/configure.zcml
```

change from:

```
  <utility
      factory=".utilities.ObservatoryController.ObservatoryController"
      provides=".interfaces.observatory_controller.IObservatoryController"
      />
```

to:

```
  <utility
      factory=".utilities.FakeObservatoryController.FakeObservatoryController"
      provides=".interfaces.observatory_controller.IObservatoryController"
      />
```

or vice versa.

You need to restart zope for this change to take effect!

Now some layout issues should be resolved, help added, side boxes removed etc...
You may want to add a script for starting the zopeinstance at boot time to `/etc/rc.d/init.d`
A sample script is in the `init.d` folder within the `Product` folder. It uses a function library called "functions" as well, your system may or may not already have this in the `init.d` folder. Add if necessary. Edit `zopectl` and `user` variables to fit your installation.
You may want to use the Plone Language Tool to force localizable parts to be in English. Choose new Plone Language Tool at the root of the Plone site in ZMI. Modify portal_languages to suit your preferences.

If you want to password protect the GUI:
Under: http://localhost:8080/guilight/manage_access
Uncheck `Anonymous` from `View`.
Do the same for: http://localhost:8080/guilight/reports/manage_access
Create a user in Plone:

http://localhost:8080/guilight/plone_control_panel
Type a valid password, reply to email etc.
Set the following permissions on the user below:
http://localhost:8080/guilight/prefs_users_overview
`member, reviewer, manager`
set permission for the reports folder (in plone) to private.

If Plone is showing an error message and the message is not explaining clearly what the problem is, a solution is to add breakpoints with the Python debugger `pdb` in the `AccessibilityObservatory` product and/or in the `DWReader2` module, and then start `zope` from the command line to investigate where it fails:

```
/opt/Plone-2.5.5/Python-2.4.4/bin/python /opt/Plone-2.5.5/lib/python/Zope2/Startup/run.py -C \
/opt/zopeinstance/etc/zope.conf
```

2.6 Managing crawls with the eiaoassess client program

When the Crawler manager, Site URL server, Sampler, WAMs and ETL server are running, the Observatory is ready to perform test runs. Test runs can be managed with the command line program `eiaoassess`. Eiaoassess communicates with the Site URL server via its web service interface to start crawls, and it communicates with the URL repository to load seed URLs.

`eiaoassess` *[-h ETLserver] [-p port] [-d] [-r test run] <-f urllist|-t table|-u URL> [-i directory]* [3]

Arguments:

-f urllist	Perform accessibility evaluations on URLs from the file urllist.
-r test run	Specify test run ID to use. (Default is EIAOXXX where XXX is the next test run ID in the URL repository, which is printed on stdout)
-t table	Perform accessibility evaluations on URLs from the table or view *table* in the URLrepository.
-u URL	Perform accessibility evaluation on the site identified by URL.
-h ETLserver	Host name of the ETL server that loads this test run. (Default is localhost).
-p port	Port number of ETL server that loads this test run. (Default is 8890).
-d	Delete the test run indicated by testrunID. This implies stopping the given crawl by removing all URLs that have been loaded for evaluation. If no test run id is given, then the system will prompt if all ongoing test runs should be stopped, and will remove all on confirmation.
-i directory	Import new URLs (*and deletes old!*) from the URL repository. Directory must contain one or more .csv-files. These files must end with .csv.

Examples:

Initially, `eiaoassess` is used to populate/seed the URL repository from .csv files. This can be done the following way:

```
eiaoassess -i /directory/with/input/csv/files
```

When the URL repository is populated, eiaoassess can be used to perform a test run, as shown in the examples below. Typically, eiaoassess is run as following to start a crawl:

```
eiaoassess -r test run -t site
```

3 Note that model is deprecated. In contrast, the variable is still part of Observatory.

where test run is an integer value indicating the current month such as 200802 and site is the table in the URL repository containing a list of all sites.

For regular monthly measurements this job will be run automatically by a cron job at given intervals.

Other possible options are:

Test the web site http://www.eiao.net

```
eiaoassess -u http://www.eiao.net -r EIAO998
```

Crawl and evaluate all URLs in the file urllist

```
eiaoassess -f urllist -r EIAO999
```

A unique test number must be used for each test run.

3 Observatory Overview

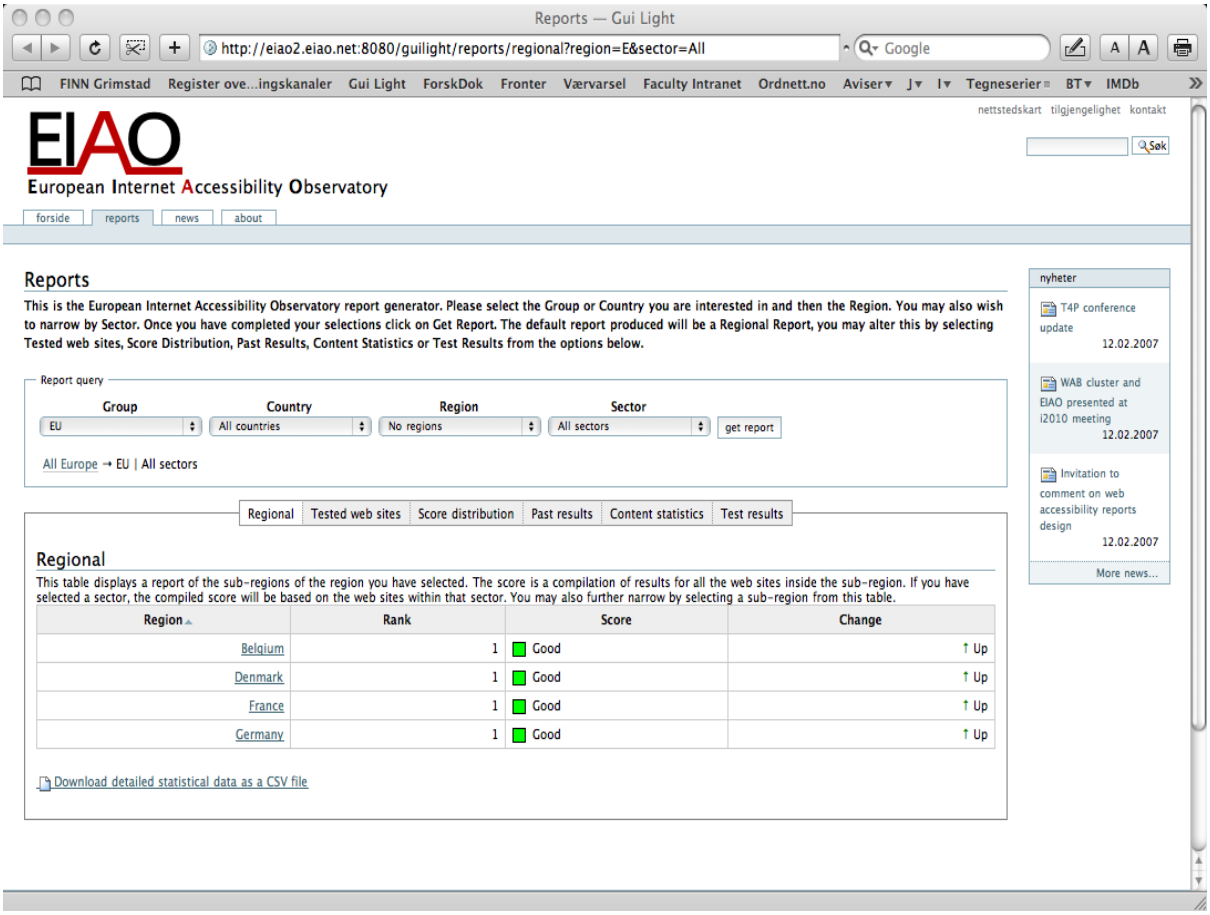


Illustration 1: EIAO on line reporting tool.

In this chapter we provide a brief overview of the how the Observatory works, including a guided tour example.

The EIAO on-line reporting tool, shown in illustration 1 [4], shows ranking lists of individual web sites, and aggregated rankings over geographical regions and business sectors. The data is presented both as scorecards and barrier indicators according to UWEM 1.2, to indicate which web sites or regions that have *good* or *bad* accessibility score.

Different sub-regions or websites belonging to a region can be compared, and history in the form of past results, and a score distribution chart can be presented for the current selection to indicate the *range* in the results. The on-line reports can also show a breakdown on the

4 Note that the example in illustration 1 is fake data.

individual UWEM indicators for a selected set of web sites, to indicate the *reason* for the given score.

In addition, the on-line reports presents content statistics about technology usage, like for instance the number of occurrences of JavaScript, JPEG, GIF or Flash per web page.

3.1 Hardware specifications

The Observatory is currently equipped with 11 servers. 6 servers run WAMs for accessibility evaluations, and 2 servers run parallel crawlers. 2 servers are dedicated to run the data warehouses. One runs the Data Warehouse being loaded and one the Data Warehouse being used. One server runs the web server. The Data Warehouse servers have each 3Tb disk space, whereas machines without much disk space, but with powerful CPUs are used as WAM and Crawler servers. To get improved WAM evaluation speed, and avoid that the crawlers and WAMs compete for the same CPU resources, we run WAM servers and crawlers on separate machines. The web server runs on a separate single-CPU server, to avoid being influenced significantly by the crawls. During a crawl, the number of crawlers and samplers are adjusted manually[5] so that the WAMs are able to keep up with the sites crawled. (I.e. Crawler machines may be reassigned as WAM machines and vice versa.)

3.2 Guided tour

In this chapter we present an example of the how the Observatory is used, as a guided tour. This includes how the URL repository is populated, how crawling and evaluation is carried out, etc. Note that this chapter is meant as a walk-through of how the Observatory typically works, for a more detailed description of each component see chapter 5. Further note that this is for reasons of clarity simplified.

(1) Categorising and scoping URLs

EIAO supports defining scope rules that provides a more fine grained mechanism for specifying which domains or parts of a domain that is part of a web site.

If we for example want to exclude everything under www.example.com/private[6] from being crawled, for the web site www.example.com, and also include the domain www.example2.com, then an XML scope pattern file needs to be made, like shown below:

5 An obvious future improvement is to do the load balancing between crawlers and WAMs automatically.

6 Note that EIAO respects the robots.txt protocol, so parts of the web site excluded via robots.txt will not be crawled.

```
<?xml version="1.0" encoding = "UTF-8"?>

<scopePatternList
  xmlns="http://www.wabcluster.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <rule
      type="exclude"
      pattern="http://www\.example\.com/private/.*"
    />

    <rule
      type="include"
      pattern="http://www\.example\.com/.*"
    />
    <rule
      type="include"
      pattern="http://www\.example2\.com/.*"
    />
</scopePatternList>
```

The scoping rules are defined in the URL repository and are inserted when new sites are inserted to the URL repository. Because of this, the scoping XML files need to be linked from the URL list .csv file used to insert new URLs to the URL repository. Each .csv URL list file contains the following fields[7]:

```
Seed URL, Site Representation, Title of Site, Source of the URL, NACE
code, Continent, Country, NUTS Category, URL to XML file with include
/ exclude patterns
```

E.g.

```
http://www.example.com, www.example.com, Example site, EIAO, 66.00,
EU, NO, NO0111,
http://www.eiao.net/scoping/scoping_for_site_www_example_com.xml
```

Inserting URLs into the URL repository from a .csv URL list file is done with:

```
eiaoassess –i /directory/with/valid/csv/files
```

If no scope file is defined, then the default scope will be used for the web site, which is:

```
include http[s]?://www\.example\.com/.*
```

7 Note that the site needs to be represented with both a seed URL and a site representation. See chapter 5.2 for further details.

(2) Site URL server

The Site URL server is automatically started upon system start as a daemon;

```
/etc/init.d/siteurlserver start
```

Prior to any crawl, the URL repository needs to be populate the URL repository. This is done by the site URL server using the command `eiaoassess`.

Inserting URLs into the URL repository from a .csv URL list file is done with:

```
eiaoassess –i /directory/with/valid/csv/files
```

Following the example from (1), the web site www.example.com will be inserted into the URL repository. This makes it possible to schedule this web site for a crawl.

The Site URL server is also responsible for scheduling the crawlers by handing out seed URLs for each site. To start a crawl, the following is needed:

```
eiaoassess –r test run –t table
```

e.g.

```
eiaoassess –r 200802 –t site
```

This will start a crawl with all sites in the URL repository (in the table site). In this example, only the web site www.example.com.

(3) Crawler

The crawlers are started automatically upon system start as a daemon;

```
/etc/init.d/crawlers start
```

This will start 50 crawlers in parallel. Here we use the web site `www.example.com` as an example. The output of the Crawler log (e.g. located in /var/log/eiao/crawler.log) would be:

```
...
Downloading URL:http://www.example.com/page2.html for site: www.example.com
Downloading URL:http://www.example.com/page1.html for site: www.example.com
Downloading URL:http://www.example.com/page3.html for site: www.example.com
...
Writing scenario: 20860104553144775519537312853047945708 5 for site
www.example.com; http://www.example.com/page1.html
Writing scenario: 13568596033618328643788938196121188038 2  for site
www.example.com; http://www.example.com/page2.html
Writing scenario: 68194615385917203052417113591653817611 for site
www.example.com; http://www.example.com/page3.html
...
```

Each web Crawler fetches the site URL to be evaluated from the site URL server, and will then perform a breadth-first search of up to 6 000 web pages, identifying web pages in the web site. The scan is optimised in subsequent scans by supporting http if-modified-since and the last-modified timestamp in the HTTP header. If if-modified-since is supported, then a single query either returns the document, or “Not modified”, which means that the URL repository already is up-to-date. If last-modified is supported, then only the HTTP header is queried, which saves bandwidth for those web pages that have not changed.

Note that in subsequent crawls all known URLs are pushed to the Crawler queue, in order to support if-modified-since or last-modified and detect any change within the web site. This is different from a breadth-first scan.

(4) URL repository

The URL repository is a database that contains tables with information about all sites and web pages identified within each web site.

The web pages above will be stored as following in the page table:

ID	URL	Site	...
20860104553144775195373128530479457085	http://www.example.com/page1.html	www.example.com	...
13568596033618328643788938196121188038 2	http://www.example.com/page2.html	www.example.com	...
681946153859172030524171135916538176 11	http://www.example.com/page3.html	www.example.com	...

Table 3: Example of URL pages written in the URL repository

(5) Sampling server

The sampling server is started automatically upon system start as as a daemon;

```
/etc/init.d/samplers start
```

When a crawl of a web site is finished, i.e. it has either identified 6 000 pages or has crawled the web site exhaustively, then the Sampler is started. It reads from the stored URLs in the URL repository.

Following our example, the Sampler would output in the crawler.log

```
Sample - 1  208601045531447755195373128530479457085:

D-value (Error Margin Reached[8], CWAM-value, Standard Deviation, Error
Margin) :

      (False, 0.3121, None, None)

      Barrier Indicator:0.3121 Lenght:133903

      http://www.example.com/page1.html


Sample - 2  135685960336183286437889381961211880382:

D-value (Error Margin Reached, CWAM-value, Standard Deviation, Error
Margin) :

      (False, 0.4103, 0.0236, 0.0024)

      Barrier Indicator:0.5482 Lenght:1374597

      http://www.example.com/page2.html


Sample - 3 681946153859172030524171135916538176l1:

D-value (Error Margin Reached, CWAM-value, Standard Deviation, Error
Margin) :

      (False, 0.4104, 0.0235, 0.0024)

      Barrier Indicator:0.25183 Lenght:845739

      http://www.example.com/page3.html

...
```

(6) WAM

The content of each downloaded (x)HTML resource in the each web page is sent to the WAM for evaluation. The WAM will autonomously download CSS files and perform cascading calculations for each element in the (x)HTML resources sent to the WAM.

The WAM will extract A-WAM indicators from the (x)HTML and referenced CSS files and combine these into B-WAM statements about barriers before the EARL report is generated from the B-WAM results. Details about how the A-WAMs and B-WAMs work can be found in D3.3.2 [D3.3.2].

A small subset of the EARL retrieved for the page http://www.example.com/page1.html (sample 1 above) is outlined below:

8 The Observatory supports different sampling policies, and sampling until a given error margin was one of the policies used. This strategy has some statistical weaknesses, so currently a fixed number of samples (600) is used and the sampling until a given error margin is deprecated and may be removed in the future.

```
<earl:TestRequirement rdf:ID="UWEM.B.10.3.2.3.CSS.DEF.1.1">
  <dc:title xml:lang="en">UWEM.B.10.3.2.3.CSS.DEF.1.1</dc:title>
  <dc:description xml:lang="en">Checkpoint 3.2: Create documents that validate to published
formal grammars. [Priority 2]</dc:description>
</earl:TestRequirement>


<eiao:SingleLocation rdf:ID="www.eiao.net.2.0.RelaxedWAM.UWEM.B.10.3.2.3.CSS.DEF.1.1-L51">
        <eiao:line>329</eiao:line>
        <eiao:column>0</eiao:column>
</eiao:SingleLocation>


      <earl:result rdf:ID="www.eiao.net.2.0.RelaxedWAM.UWEM.B.10.3.2.3.CSS.DEF.1.1-V51">
           <earl:validity rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-strawman#fail"/>
           <eiao:barrierIndicator>1</eiao:barrierIndicator>
      </earl:result>


<earl:Assertion rdf:ID="www.eiao.net.2.0.RelaxedWAM.UWEM.B.10.3.2.3.CSS.DEF.1.1-A51">
        <earl:assertedBy rdf:resource="http://www.eiao.net/2.0/RelaxedWAM/"/>
        <earl:subject rdf:resource="http://www.example.org/"/>
        <earl:requirement rdf:resource="#UWEM.B.10.3.2.3.CSS.DEF.1.1"/>
        <earl:mode rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-strawman#automatic"/>
        <earl:result rdf:resource="#www.eiao.net.2.0.RelaxedWAM.UWEM.B.10.3.2.3.CSS.DEF.1.1-
V51"/>
        <eiao:singleLocation
rdf:resource="#www.eiao.net.2.0.RelaxedWAM.UWEM.B.10.3.2.3.CSS.DEF.1.1-L51"/>
</earl:Assertion>
```

This EARL shows that on line 329 column 0 UWEM Test `UWEM.B.10.3.2.3.CSS.DEF.1.1` failed, which indicates a barrier at this location.[9]

9 In addition to the EARL, the EIAO specific eiao:barrierIndicator shows the barrier indicator value which can be a real number between 0 and 1, and the eiao:SingleLocation RDF class shows the location in the document where the problem occurred. The EIAO RDF is also extended with the possibility to pass other document data (e.g. content type of referenced resources) to the Data Warehouse.

The B-WAM results[10] identified for this web page were:[11]

BWAM ID	Description	Result (barrier indicator)
UWEM.B.10.1.1.3.HTML.DEF.6.1	Checkpoint 1.1: Provide a text equivalent for every non-text element	pass (0)
UWEM.B.10.3.2.3.HTML.DEF.1.1	Checkpoint 3.2: Create documents that validate to published formal grammars. (HTML)	fail (1)
UWEM.B.10.11.1.3.HTML.DEF.1.1	Use W3C technologies when they are available and appropriate for a task and use the latest versions when supported.	fail (1)
UWEM.B.10.3.2.3.CSS.DEF.1.1	Checkpoint 3.2: Create documents that validate to published formal grammars. CSS, location (329,0)	fail (1)
UWEM.B.10.3.2.3.CSS.DEF.1.1	Checkpoint 3.2: Create documents that validate to published formal grammars. CSS, location (341,0)	fail (1)
UWEM.B.10.3.2.3.HTML.DEF.2.1	Checkpoint 3.2: Create documents that validate to published formal grammars.	fail (1)
UWEM.B.10.7.2.3.HTML.DEF.1.1	Checkpoint 7.2: Until user agents allow users to control blinking, avoid causing content to blink (i.e., change presentation at a regular rate, such as turning on and off). (HTML)	pass (0)
UWEM.B.10.7.2.3.CSS.DEF.2.1	Checkpoint 7.2: Until user agents allow users to control blinking, avoid causing content to blink (i.e., change presentation at a regular rate, such as turning on and off). (CSS)	pass (0)
UWEM.B.10.7.3.3.HTML.DEF.1.1	Checkpoint 7.3: Until user agents allow users to freeze moving content, avoid movement in pages. [Priority 2]	pass (0)

In addition, the following content statistics data for the evaluated web page was identified and sent in the EARL/RDF report:

Name	Type	Value
EIAO.TECH.1.2	technology	text/html
EIAO.LANG.1.2	language	nb-NO

Both pass and fail results have been shown, since the UWEM 1.2 aggregation algorithm needs both. The EARL reports for all web pages are then stored in an RDF graph to be read by the ETL along with content statistics data.

10 Note that this is barriers for the complete web page - both the (X)HTML and CSS-files.
11 Both PASS and FAIL results are returned via the EARL report.

(7) ETL

The ETL server is automatically started upon system start as a daemon;

```
/etc/init.d/server start
```

Whenever the Sampler has successfully sampled a site, the RDF database is sent to the ETL server for loading into the Data Warehouse.

The ETL server handles requests from the Sampler to load RDF databases with site results, and calls the ETL loader to load the results into the Data Warehouse. For efficiency reasons, this process is done in a pipeline. The ETL server allows one thread to perform the Extract part of the ETL by caching the RDF database to memory[12], while the other thread in parallel performs the Transfer and Load into the Data Warehouse of another RDF database. This is done to maximize the utilisation of the ETL[13].

```
Following our example:
      Currently handling website:
            http://www.example.com
      Currently handling resource
            http://www.example.com/page1.html
      Currently handling resource
            http://www.example.com/page2.html
      Currently handling resource
            http://www.example.com/page3.html
      ...
      Time spent in total: 22.75 secs.  Time spent on reading RDF: 8.80 secs.
      Thu Mar 22 11:31:47 2007 Finished loading data from the RDF database
      Added 7883 facts
```

(8) Data Warehouse

The Data Warehouse contains all the basic accessibility data, and also all precomputed aggregates that are needed to present the data in the user interface. More information about the ETL is described in D6.1.1.1 Data Warehouse functional specification.

CWAM aggregation, page level

12 The amount of memory needed for loading an RDF database in memory is limited, since the RDF database is limited to the samples from one web site.

13 The ETL is not yet parallelised, so it is important for Data Warehouse load performance that the Extract part can run in parallel with the Transform and Load parts of the ETL.

The CWAM aggregation is based on N_p - the total number applications of all tests within page *p* and B_p - the total number of "FAIL" results from all tests in test set *T* within page *p.*

In the example page from www.example.com, the number of applicable tests $N_p = 9$ and the number of failed tests is $B_p = 5$. The UWEM 1.2 score for this page is therefore:

$$f(p) = \frac{B_p}{N_p} = \frac{5}{9} \approx 0.56$$

CWAM aggregation, site level
The UWEM indicator on site level is calculated by taking the ratio of all failed instances over all applicable tests (pass and fail):

For example, if sample s consists of three pages with values

$f(p_1) = \frac{5}{9}$, $f(p_2) = \frac{37}{86}$, and $f(p_3) = \frac{26}{83}$,

then the UWEM score is $F(s) = \frac{5+37+26}{9+86+83} = \frac{68}{178} \approx 0.38$.

Aggregation of a region

The results for a region consisting of *n* web sites is calculated by taking the average of the individual web site results:[14]

$$F(R) = \frac{1}{\text{size}(R)} \sum_{s \in R} F(s)$$

If we assume to have the region Oslo, including 3 regions; www.example.com, www.anotherexample.com and www.thirdexample.com:

$$F(R) = \frac{1}{3}(0.38 + 0.17 + 0.25) = 0.27$$

14 We have had a discussion of only presenting results when a minimum number of sites are part of a region.

Region (NUTS3 level)	*Site*	*F*
Oslo	www.example.com	0.38
Oslo	www.anotherexample.com	0.17
Oslo	www.thirdexample.com	0.25
Oslo		**0.27**

(9) On-line reports

The EIAO web based user interface would compare the score value to the cut-off values in the UWEM 1.2 scorecard scale:

Scorecard		*Score value*
Letter	*Colour*	*F(s)*
A		$F(s) = 0$
B		$0 < F(s) \leq 0.25$
C		$0.25 < F(s) \leq 0.5$
D		$0.5 < F(s) \leq 0.75$
E		$0.75 < F(s) \leq 1$

Table 4: UWEM 1.2 scorecard scale.

The conclusion is that the score of 0.27 for Oslo in this example represents a score letter of C and a score colour of yellow.

3.3 Pre-caching of on-line reports

To perform precaching of the on-line reports, which improves the interactive responsiveness after the Data Warehouse has been loaded, the following steps are required. This makes sure that both the old cache results are 'removed' from the on-line reports and that new results are presented.

1. Delete all /tmp/*.dmp files. These files contain cache results from earlier requests to the Data Warehouse.

2. Restart zope.

```
cd /var/lib/zope/bin
/etc/init.d/zope restart
```

3. Make sure postgres is running;

```
ps aux | grep postmaster
```

4. Empty the `dwtime.log`.

```
cd /var/log/eiao
mv dwtime.log dwtime_old.log
touch dwtime.log
chown zope:zope dwtime.log
```

5. Perform pre-caching for different NACE codes by running:
   python performcache.py <URL_TO_On-line reporting tool>
   <LIST_OF_NACE_CODES>
   E.g:

```
python performcache.py http://eiao2.eiao.net:8080/guilight/
ALL,47.91,53.10,58.13,60.10
```

   Remember to run in a screen (or similar) since this pre-caching can have a long duration

6. During the pre-caching, the following should be continuously monitored to make sure everything is running correctly;
   - Make sure that data becomes continuously available in the /tmp/*.dmp files
   - Make sure that requests are sent to the Data Warehouse. All requests are presented in /var/lib/pgsql/data/pg_log/
   - Make sure that zope and postgres are up and running;
     ps aux | grep zope
     ps aux | grep postmaster
   - Make sure that zope and postgres are getting requests. Typically, zope + postmaster runs at 100% cpu. This could be checked with `top`.

7. Avoid a lot of visits to the on-line reports during the caching process. Currently Zope can only handle four requests simultaneously. This means that at most four visits can be done to the on-line reports at the same time which is a limitation before the on-line reports is completely cached because it can easily come a situation where requests take a long time. This might further make the on-line reports halt because of too many requests. When the pre-caching is finished, visit the on-line reports as much as you like.

4 EIAO architecture

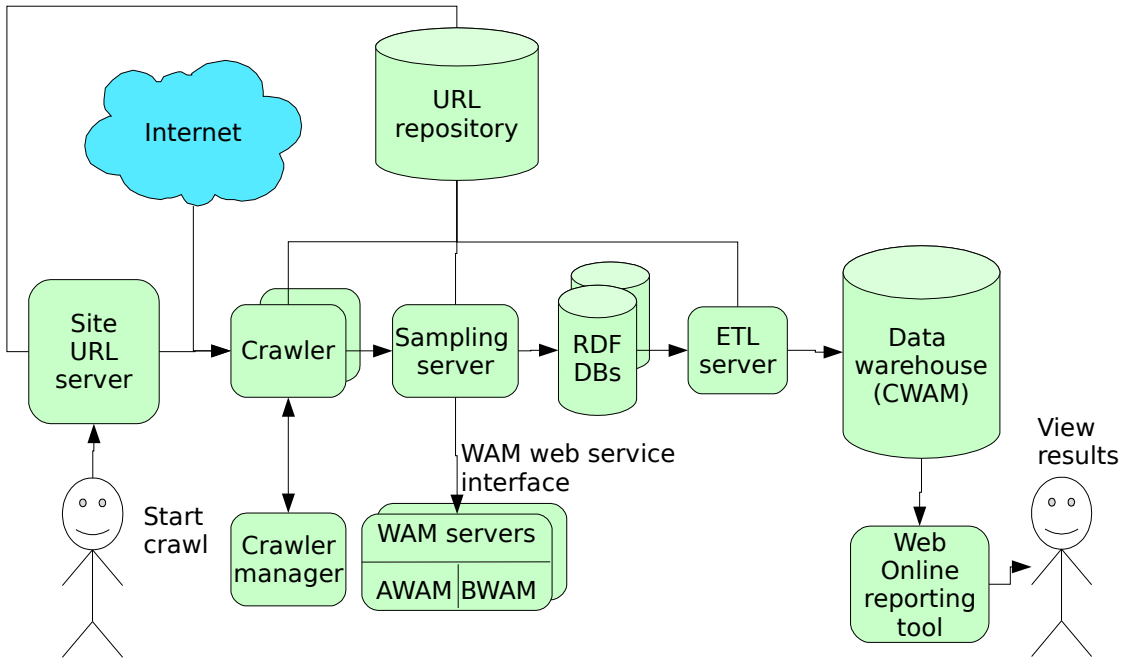


Illustration 2: Parallelised Observatory architecture of EIAO version 2.2.

In this section we describe briefly the Observatory's components. The pipeline of operations when evaluating web pages is indicated by the arrows in the Observatory architecture illustration.

4.1 Site URL server

The Observatory consists of a site URL server, that delivers URLs for web sites to be evaluated by a set of parallel web Crawler processes. The operator will load the site URL server with a list of URLs, fetched either from a file or from the URL repository by using the command line function: `eiaoassess`.

4.2 URL repository

The URL repository is a database that contains a table with all sites, and flags indicating the state of each site (enabled, disabled etc.) In addition, there is a table for each web page found within a web site, with all the URLs found within that web page. There is also a table for newly identified domain URLs.

4.3 Crawler manager

The `crawlers` command is the main Crawler manager that starts all Crawler processes on a Crawler machine, and verifies that all Crawler processes are running and progressing. There is one Crawler manager for each machine that runs crawlers.

4.4 Crawler

Each web Crawler operates on a single web site at a time. It starts by fetching the site URL to be evaluated from the site URL server, and will then perform a breadth-first search of up to 6 000 web pages, to harvest links to web pages and store these in the URL repository.

The Crawler also performs scoping checks and checks for robots.txt, so that only URLs that are within scope are added to the URL repository.

4.5 Sampling server

The sampling server is separated from the Crawler server as a separate service. After the Crawler has identified 6 000 pages or complete web site, the web site is passed on to the sampling server responsible for managing individual samplers.

4.5.1 Sampler

The sampler extracts a uniform random sample of 600 pages[15] from the set of pages identified and stored in the URL repository. Each of the selected URLs will then be downloaded and sent to the WAM for evaluation. The WAM returns an EARL report to the Sampler, which then calculates the accessibility score of the site *F(s)*.

The results are then stored in the RDF database for this web site. The Sampler requires at least 30 samples (i.e. 30 pages identified) to send the result to the ETL for loading[16]. Web sites with less pages than this will need to be inspected manually to identify if the site has been crawled exhaustively or if there are problems with forwarding, scoping or unsupported technologies (e.g. Flash or extensive use of JavaScript[17]) or other problems. When all web pages have been evaluated, the RDF database is sent to the ETL load server for loading into the Data Warehouse. The sampling server will then continue with the next web site to sample. The sampling server is multithreaded, and supports sampling different web sites in parallel[18].

15 The sampling server will sample 600 samples without replacement. If there are less than 600 pages in the web site, then the entire web site will be sampled.

16 Note that sites where less than 30 pages have been identified by the Crawler, will, for performance reasons, be discarded by the Sampler without sending requests to the Sampler. For such sites it is clear that the sample cannot reach the minimum required evaluation of 30 pages. It is possible to mark a site that has been verified to have less than 30 pages for sampling by setting the smallsite flag in the site table of the URL repository. If the smallsite flag is set, then the site will be sent for evaluation regardless of the number of pages crawled.

17 The Crawler is to some extent able to parse JavaScript, to identify links, but it does not have complete JavaScript support.

4.6 WAM server

The WAM server is a multi-threaded server that receives SOAP requests [SOAP] to evaluate a web page, and returns an EARL report [EARL] with the evaluation result. (x)HTML processing is based on the Relaxed HTML validator, which has a Schematron /RNG [ISO19757-3] processing engine.

(x)HTML documents are evaluated using the WAM Schematron rules. See D3.2.2 WAM specification for more information about them. HTML validation is done using Xalan [Xalan-J] in validation mode.

The WAM handles CSS stylesheets in a similar way as a web browser, i.e. by parsing the (x)HTML document for references to CSS and perform CSS cascading calculations for each HTML element, to decide which CSS rules that apply to the HTML element. Only applicable CSS rules are checked by the AWAM module, presuming a default media type of screen. This means that e.g. only blinking text that would be visible on a web browser would count as a barrier indicator. CSS validation is done at the same time as the CSS rules are parsed by the Batik CSS SAC parser[BATIK].

The results from the schematron evaluation, validation and CSS evaluation are sent in a common AWAM data structure to the BWAMs for UWEM barrier indicator computations and to extract content statistics from the web page (e.g. content type). The BWAM results are currently either 0, indicating no barrier indication and 1 indicating barrier indication. The BWAM results are then converted to an EARL report, which is returned to the Sampler.

4.7 ETL server

Whenever the sampling server has successfully sampled a site, the RDF database and corresponding parameters are sent to the ETL server, which starts an ETL load to load the evaluation data into the Data Warehouse.

The ETL server handles requests from the Sampler to load RDF databases with site results, and further calls the ETL to load the results into the Data Warehouse.

For efficiency reasons, this process is done in a pipeline. The ETL server allows two parallel processes to run the Extract part that reads the RDF database simultaneously. One Extract process caches the RDF database to memory[19], while the other is allowed to extract data and load this into the Data Warehouse. This is done to minimize the limitation of the Data Warehouse that the Transform and Load parts only supports one ETL to insert at the same time.

The reason why the Transform and Load parts have not been parallelised, is that the ETL process uses data already stored in the Data Warehouse in the transform- and load-part. Example of such a transformation/load is:

```
If mimetype not in in Data Warehouse:
     insert mimetype.
```

18 In order to utilize the CPU of the WAMs, five sampling server threads are run in parallel on one the sampling server machine.

19 The amount of memory needed for loading an RDF database in memory is limited, since the RDF database is limited to the samples from one web site.

Previously, the extraction-part was the main bottle-neck of the ETL. However, reading RDF-graphs is currently not the limiting factor of the ETL. For further details on this, see [TestReport].

4.7.1 ETL

The ETL performs the Extract, Transform and Load operations into the Data Warehouse, and adds indexes and materialized views after the load of a web site has finished. More information about the ETL is described in D6.1.1.1 Data Warehouse functional specification.

4.8 Data Warehouse

The Data Warehouse contains all the basic accessibility data, and also all precomputed aggregates that are needed to present the data in the user interface. More information about the Data Warehouse is described in D6.1.1.1 Data Warehouse functional specification.

4.9 On-line reports

The Observatory user interface is an extension of a content management system (Plone) that reads data from the Data Warehouse, and presents these as regional, sectoral or other reports using the UWEM scorecard scale. The on-line reports are separated into a model, that connects to the Data Warehouse, a view that presents the results, and a controller part that implements ways of interacting with the viewed results.

5 Details on Observatory parts

This section describes implementation details about each component of the EIAO Observatory. It is mostly useful for developers wanting to participate in further development of the Observatory or developers wanting to do their own adaptations.

5.1 SiteURLServer - Site URL server

Location: robacc/SiteURLServer/siteurlserver

The Site URL server consists of a web service that listens to port 8889 by default, and provides functions to load and manage the queue of URLs provided to the crawlers. The `eiaoassess` client will add or remove site tuples to the queue of URLs to be evaluated. Each site in the queue is represented by a tuple that consist of:

`(siteurl, test run, etlserver, scheduleCount`[20]`, timeout`[21]`)`

The test run parameter is used to indicate which test run this site belongs to, so that the ETL service knows which test run to load.
The Site URL server maintains a list of test run batches (Queues) where new URLs can be inserted into any new or ongoing (but not finished) testruns. A list of queues is useful to simplify the operation of removing individual test runs with the -d option[22].

The command line interface `eiaoassess` is a client interface to the Site URL servers web services. URLs to be evaluated can be given on the command line (-u option), as a list of urls in a file (-f option) and as a table/view name in the URL repository (-t option). The loading of URLs consist of pushing site tuples to the queue of URLs to be evaluated. Removing a test run will involve removing the given test run from the given queue. If no test run is specified, then all test runs are removed. (Requires confirmation).

20 scheduleCount is deprecated after the exhaustive scan approach was introduced. It may be removed in future versions of the Observatory. Currently, manual investigation is performed for web sites that are not being crawled successfully, and the web site is reinserted with eiaoassess when the issue that prevented successful crawling (e.g. forwarding issue) has been solved.

21 timeout is deprecated after the exhaustive scan approach was introduced. It may be removed in future versions of the Observatory.

22 The Python thread safe Queue class does not support slicing or other operations in the middle of the queue. It is therefore regarded as a simpler and cleaner approach to maintain one queue per test run than to maintain all test runs in one queue. In addition, this approach will facilitate more advanced scheduling in the future, so that other scheduling algorithms, like round-robin, or priority based scheduling between test runs easily can be introduced. One use case could be having a highly prioritised interactive test run for on-line site measurements of single web sites, that could run in parallel with the monthly crawls, so that we in the future could provide an on-line site assessment service in addition to the monthly crawls.

The site URL servers web services interface consists of the following functions:

```
   def addURLs(self, test run, rdfmodel, urllist=[], etlserver=”localhost”,
               port=8889):
        """add a list of URLs to the queue for the given test run,
           and reset the scheduleCount to 0
        """

   def addURLsFromTable(self, test run, rdfmodel, etlserver='localhost',
                  siteurltable='Site'):
        """add another batch of site urls the table given by urltable
           in the URL repository
        """

   def removeTestRun(self, test run=ALL):

        """Remove all URLs for a given test run id. Default is ALL
           (However this needs confirmation in the eiaoassess client.)
        """

    def getSiteURL(self):
        """return the next URL tuple (test run,URL) from the current queue
           and increase the scheduling counter
        """

    def putSiteURL(self, sitectrl, test run, rdfmodel,
etlserver=”localhost”,
                   schedulecount=0):
        """put a site URL tuple on the queue
        """

    def addSiteURLtoTestrun(self, siteurl, test run, rdfmodel,
                  etlserver="localhost", schedulecount=0)
        """Add a new site URL to a test run
        """
```

Both loading of a single URL and loading of a list of URLs from a file is done via the `addUrls()/addURLsFromTable()` call from the `eiaoassess` client. The `addURLs()` and `addURLsFromTable()` functions will send a `testRun()` message to the ETL server, to notify it about the amount of URLs in the new test run. `getSiteURL()` is used by the crawlers to request a new site URL to crawl and evaluate and `putSiteURL()` is used by the crawlers if they for some reason have to abort the evaluation of the web site, so that another crawler later can evaluate this web site.

5.1.1 Starting a test run

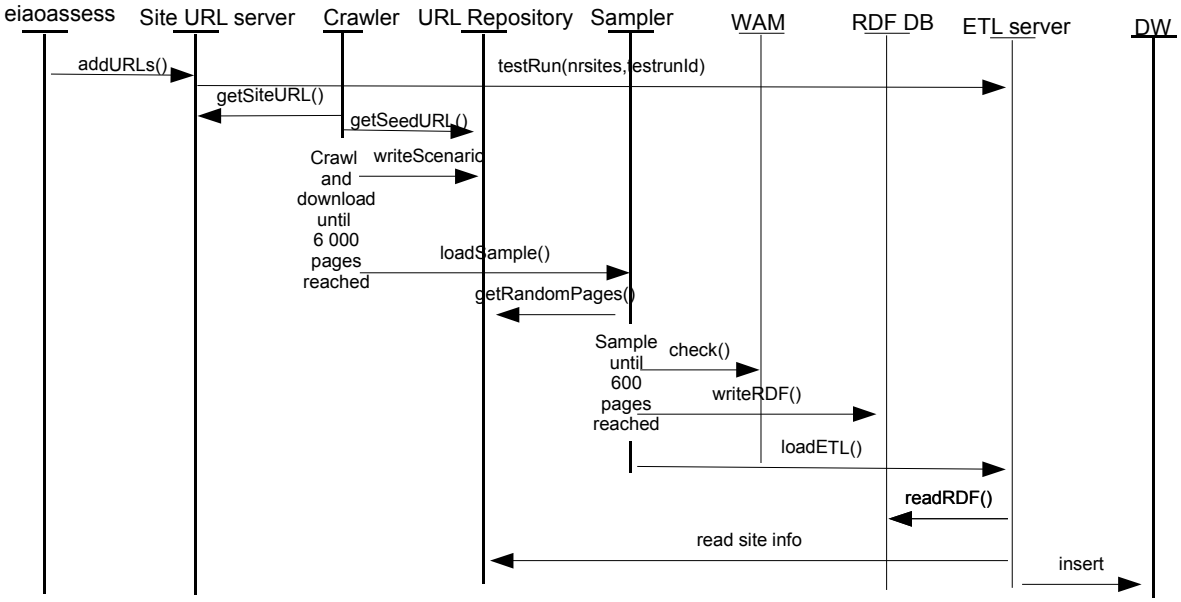


Illustration 3: Simplified sequence diagram showing normal Observatory functionality and interaction with the site URL server. Note that all calls are synchronous calls that return a value and the arrow indicates caller/callee.

When a list of URLs to crawl has been loaded by `eiaoassess`, then the URLs will be added to the Site URL server via the `addURLs()` call, or alternatively by `addURLsFromTable()` to start a test run on URLs stored in a table in the URL repository. The Site URL server will then inform the ETL server about the number of sites to expect in the test run with the `testRun(nrsites,testrunid)` call.

Crawlers will call the Site URL server for the next web site to crawl by calling the `getSiteURL()` method in the site URL server. The crawler will then get the seed URLs from the URL repository by using the `getSeedURL()` method of the URL repository, and the crawler will crawl and download up to 6 000 pages from this web site, and write information about web pages identified with the `writeScenario()` method. When 6 000 web pages have been crawled, or all identified web pages if the web site contains less than 6 000 pages, the crawler will inform the Sampler server that the web site can be sampled by calling `loadSample( )` with the web site to sample as argument.

The Sampler will then get a random set of 600[23] web pages, or all web pages if the URL repository contains less than 600 pages, from the URL repository by calling the method `getRandomPages()`. The sampler will then iterate through all web page URLs, download the web page, send the web page to a WAM for evaluation using its `check()` method and the resulting EARL is then inserted into the RDF database together with some other

23 Note that more web pages are extracted from the URL repository if some pages are unable to evaluate due to e.g. they are no longer downloadable. However, only 600 pages are at most sampled.

parameters like HTTP header parameters. Further more, the sampler makes sure the RDF graph stored in the database is a complete connected graph. This makes it possible to extract EARL from each individual page from the ETL.

When all samples have been acquired, the sampler calls the method loadETL() on the ETL server with the RDF database to load as argument. The ETL server will then read the RDF graph from the RDF database in to memory using the readRDF() method of the RDF database, and it will then start an ETL instance which loads data from the RDF database and NUTS/NACE categories from the URL repository into the Data Warehouse.

5.1.2 Aborting a test run

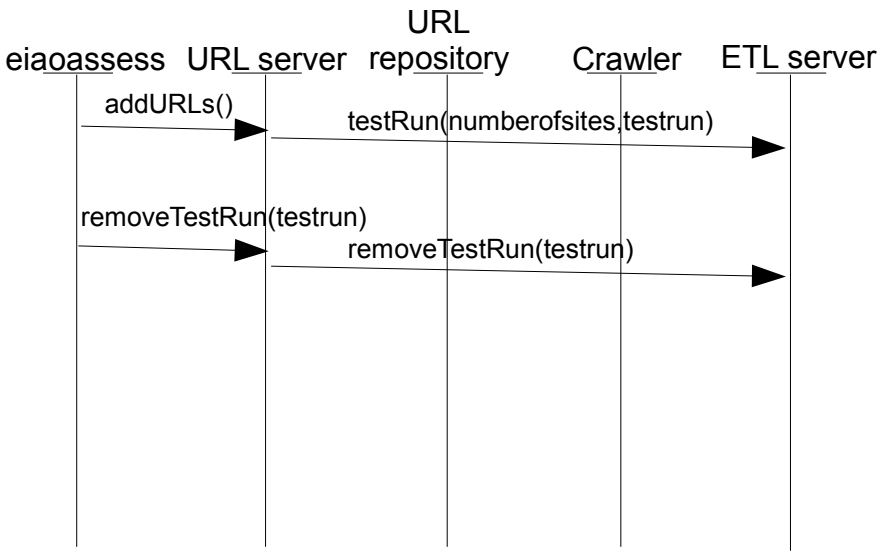


Figure 5.1: Simplified sequence diagram showing removal of a test run. All calls are synchronous blocking SOAP calls, and the arrow indicates caller/callee.

If the test run is removed, then the ETL server also needs to be notified by sending a removeTestRun() message to the ETL server, so that it can remove the load and fail counters for the given test run.

5.1.3 Persistent state

The Site URL server stores its state whenever the internal queues change, to be able to restart where it left off if the Site URL server should terminate abnormally. The state is stored in the following files:

File	Data structure
/tmp/siteurlqueuecache.dmp	Site URL queue
/tmp/testruncache.dmp	Current test run

5.1.4 Eiaoassess command

Location: robacc/SiteURLServer/eiaoassess

Eiaoassess is used to manage test runs in the site URL server and to load URLs into the URL repository. It uses the web services interface published by the site URL server to manipulate the test run data. These functions are described in section 5.1.

The -i option for importing URLs operates directly towards the URL repository. It uses the URL repository function addInitialURLs() to add URLs.

NOTE: The -i option is destructive and will delete the URLs in the URL repository before new URLs are added. Remember to take backup of the URL repository before using this option!

5.1.5 Site URL server modules

Location: robacc/SiteURLServer

Module	***Description***
eiaoassess	Commend line module for interacting with the Site URL server
siteurlserver	Service for managing crawls

5.2 URL_repository - URL repository

Main module: robacc/URL_repository/urlrep.py

In the following chapter we describe the URL repository which stores information about all web sites that are evaluated, how they are categorised, scoping and information about individual web pages and sets of pages in a frameset[24].

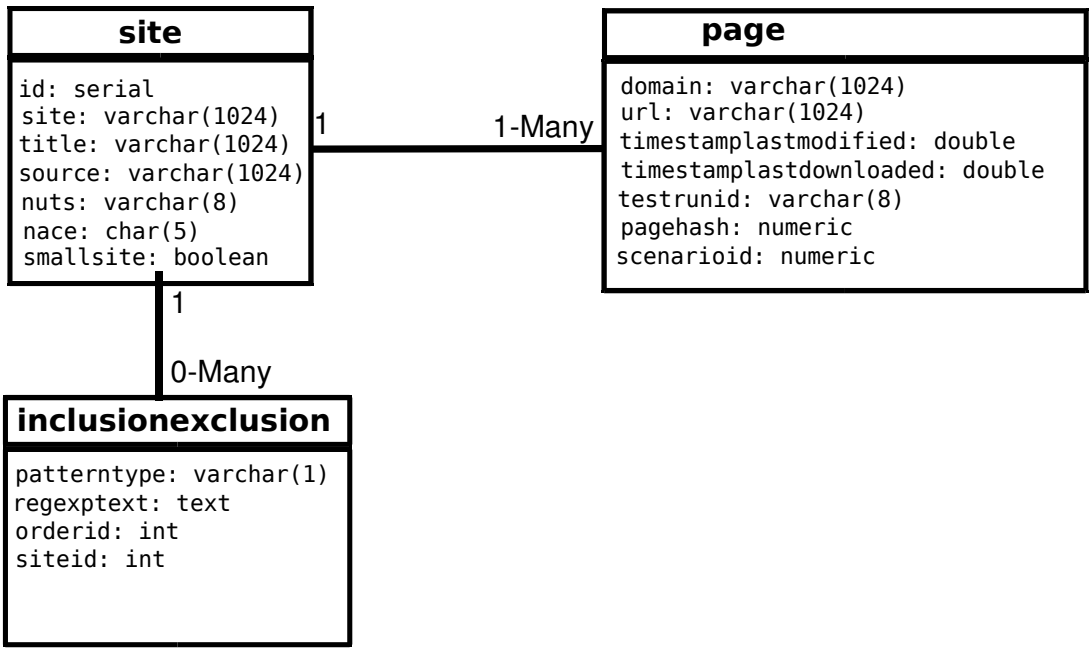


Figure 5.2: Overview of URL repository tables

The URL respository consists mainly of a site table that stores basic information about a given web site and how it is categorised into NUTS and NACE categories. A web site may have 0 or more scoping rules. Each web site may have one or more web pages. The page table also stores information about timestamps to keep track of last-modified and HTTP if-modified-since handling and it contains a testrunid to store information about in which testrun a given web page was identified, and a scenarioid which is used to indicate which web pages that belong together in a frameset.

An example of how information about a web site could be stored in the URL repository, assuming there are two pages with id 1 and id 2 for the web site www.example.com is shown in Table 5.

24 Note that for reasons of clarity, only the most important tables, and main parts of these tables are presented.

page table:

Site	URL	...	ScenarioID
www.example.com	http://www.example.com/index2.html	...	1
www.example.com	http://www.example.com/index.html	...	2
www.example.com	http://www.example.com/frame1.html	...	2
www.example.com	http://www.example.com/frame2.html	...	2

Table 5: Example of pages stored in the URL repository

In the above example, web page with scenarioID 1, is a “traditional” web page without frames[25]. Web page with scenarioID 2 is a frame page with the main frame-set page index.html, and two corresponding pages; frame1.html and frame2.html as pages within the frame [26]. This means that we keep all information about downloaded pages for all test runs. We can further partition the table on test run id.

The pages shown in in table 5 are organised by site, in this case represented by the domain www.example.com. Note that the site representation is not in all cases equal to domain, since one domain may include more than one site. From the point of view of the URL repository, the only restriction for organising pages correctly into sites is that each site field is unique for each particular site. However, for the ETL and Data Warehouse the site description must represent the site itself. This also makes the site representation more readable. Note that for web sites scoped only by domain, simply organising the sites according to domain is sufficient.

In contrast, if one domain is scoped into several sites then another site representation is needed. For such sites it is required that the site description is represented by a URL corresponding to the site, typically the main seed URL of the site. Note that this description is how the site will be presented in the on-line reporting tool.

For reason of clarity, we present an example.

Assume that you have two web sites:
(1) http://www.example.com/site1 and
(2) http://www.example.com/site2.

The above web sites are within the same domain. Thus representing these sites by domain only, will cause problems since it would not be possible to separate these two web sites from each other in the URL repository. It is therefor required that these site descriptions are represented by the URL corresponding to the site. In this case, site (1) should be represented by www.example.com/site1 and (2) by www.example.com/site2. Example of the page table with the above two sites can be seen in table 6[27].

25 Note that CSS information is not stored in the URL repository, in contrast to previous versions of the Observatory.
26 Note that if a page within a frame, e.g. frame1.html in this example, is available in the web site outside of the frame page index.html, it will additionally be stored with a separate ID.
27 The Site table, in addition to the Page table shown in this example, has the same site representation.

Page table for two web sites:

Site	URL	...	ID
www.example.com/site1	http://www.example.com/site1/page1.html		1
www.example.com/site1	http://www.example.com/site1/page2.html		2
www.example.com/site2	http://www.example.com/site2/page1.html		3
www.example.com/site2	http://www.example.com/site2/page2.html		4

Table 6: Example of pages stored in the URL repository showing two sites defined within the same domain by using scoping rules.

5.2.1 Scoping rules (include / exclude patterns)

The scope of a web site is specified as a set of scoping rules which consist of include/exclude patterns. Each pattern is defined as regular expression [REGEXP] ether as an include or exclude pattern. Include patterns are indicated by a '+' in the patterntype field and exclude patterns are indicated by a '-' in the patterntype field of the inclusionexclusion table. The regular expression for the rule is stored in the field regexptext and the orderid is the order of application of this rule. Inclusionexclusion has a 1-many relationship with the site table – one site can have one or more scope pattern rules. The Crawler will only follow URLs that are within the scope of the site.

The default scope of a web site is to stay within the domain of the home page URL. The include pattern of such a scope would be defined as following:

http[s]?://www\.example\.com/.*

The Crawler would then follow URLs that are within the domain www.example.com and reject URLs that are outside of this domain.

5.2.2 Example of scope rules in the URL repository

The scope of a web site is defined as a set of patterns that will be applied to each URL the Crawler detects. A URL that matches an including scoping rule, will be crawled. In contrast a URL that matches an excluding scoping rule, will be discarded.
These rules need to be defined in the order from the most specific to the most general.

As an example, the scope of a web site could be defined as following; The scope of the site www.example.com includes every URL from the site http://www.example.com/site1/ except http://www.example.com/site1/admin/. However, the parts of the sites under http://www.example.com/site1/admin/private/ and http://www.example.com/site1/admin/public/ should be included. Furthermore, any URL with the parameters ?type=notsosecret&isitsecret=no should be also be included.

This gives the following include/exclude patterns in the URL repository:[28]

inclusionexclusion table:

Siteid	*orderid*	*type*	*regexptext*
1	1	+ (Include)	http://www\.example\.com/site1/admin/private/.*
1	2	+ (Include)	http://www\.example\.com/site1/admin/public/.*
1	3	+ (Include)	.*?type\=notsosecret\&isitsecret\=no.*
1	4	- (Exclude)	http://www\.example\.com/site1/admin/.*
1	5	+ (Include)	http://www\.example\.com/site1/.*

Table 7: Include/exclude patterns in URL repository.

These patterns, in the order above, are converted from XML include/exclude pattern files according to the schema in appendix A. An example of such as schema is available appendix B.

5.2.3 URL repository modules

Location: robacc/URL_repository

Module	*Description*
urlrep.py	URL repository module. Used as mapping between python and the postgreSQL URL repository
tables.sql	SQL definition of URL repository relational database.
nuts3.csv	Mapping between locations and NUTS3.
createeiaourlrep10.py	Create URL repository database.
deleteeiaourlrep10.py	Delete URL repository database.
parsexmlinclusionexclusion.py	Parse XML include/exclude scoping patterns.
URLChecker.py	Module for validating values connected to inserted URLs, such as NUTS/NACE
notetlserver.py	Dummy ETL server used for testing

28 Note that several fields that are available in the URL repository have been excluded from this example. The reason for this is to make the example of scoping rules as clear as possible.

Module	*Description*
notwebserver.py	Dummy web server used for testing
setup.py	Installation module
siteurlclient.py	Client for a direct interaction with the URL repository
test.py	Testing the URL repository
timetest.py	Testing performance of the URL repository
urlreperror.py	Module for raising URL repository related exceptions
__init__.py	Package definition.

5.3 Crawlers - Crawler manager

The `crawlers` command is the main Crawler manager that starts all Crawler processes on a Crawler machine, and verifies that all Crawler processes are running. It also implements the watchdog restart functionality, to restart crawlers that have not reported progress within the watchdog timeout period. The watchdog functionality is described in section 5.3.1. It also handles restart of crawlers due to excessive memory usage. This is described in section 5.3.2.

5.3.1 Watchdog functionality for hanging crawlers

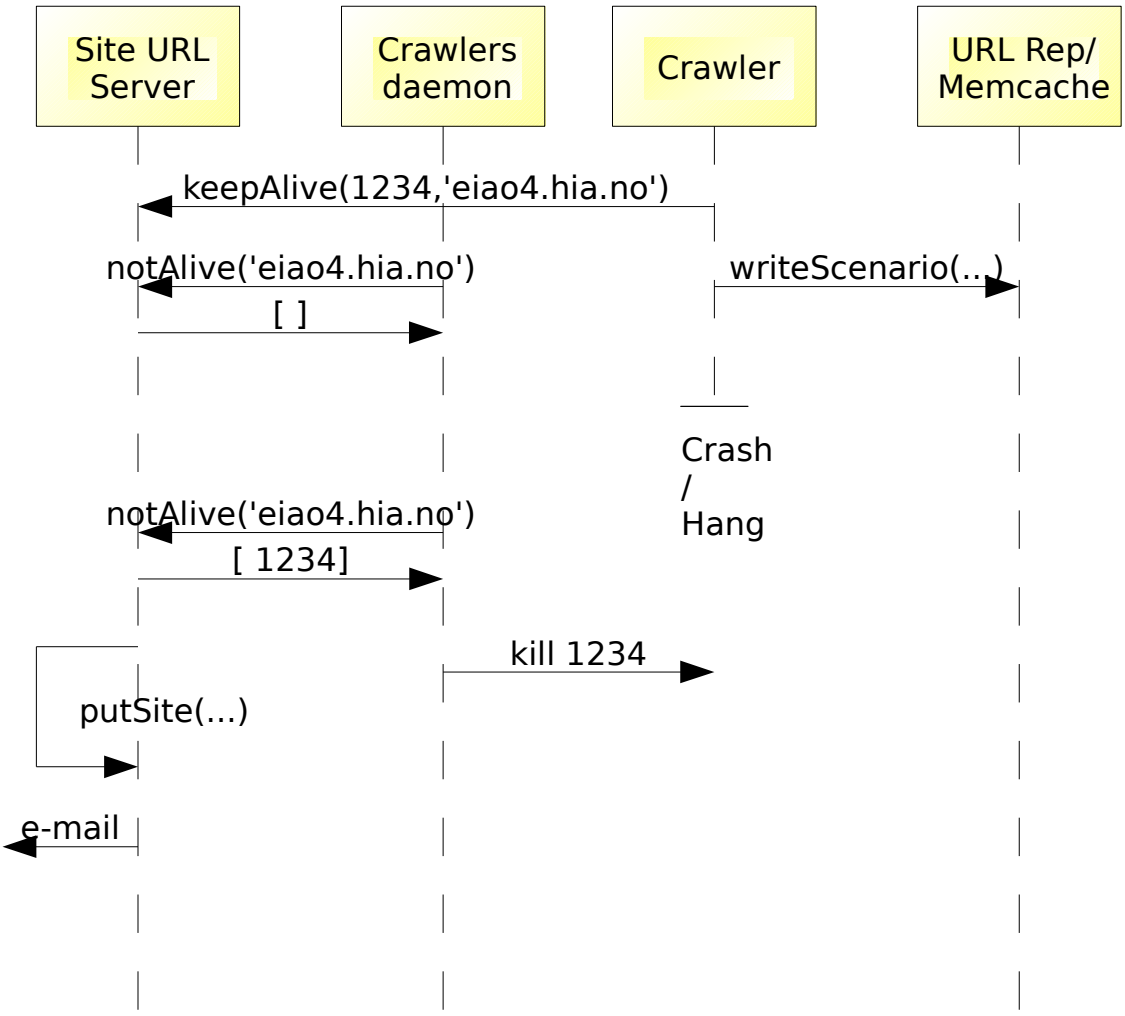


Figure 5.3: Example of the watchdog terminating a hanging Crawler

In order to deal with crawlers that are hanging, a watchdog functionality has been implemented. This includes sending `keepAlive()` messages at regular intervals from each running Crawler. Any Crawler that for some time has not sent a `keepAlive()` message, is assumed to be hanging and will be restarted. Note that this functionality is only useful when the errors making the Crawler hang are **not** deterministic; only arbitrary errors, such as arbitrary dead locks, may be avoided with this functionality. The `memcached` functionality, described in 5.4.6, allows the Crawler to restart without loosing state information.

The `keepAlive()` messages are sent from the Crawler to the Site URL server after a page has been downloaded given that there has been at least 60 seconds since last time `keepAlive()` was sent. The parameters of these massages are presented in Table 8.

Furthermore, the crawlers daemon, responsible for starting and stopping crawlers, polls the Site URL Server at regular intervals with a `notAlive()` message. Its parameters are presented in Table 9. The Site URL Server responds with PIDs of crawlers that have not sent `keepAlive()` for at least 5 minutes. Note that the large difference between 5 minutes and 60 seconds is to allow temporary delays without killing a Crawler. E.g. a crawl can use more than 60 seconds downloading a page without being killed. The 60 seconds delay is configurable in the system configuration file initial.rdf via the RDF property `maxkeepalive` and the 5 minutes delay can be configured via the RDF property `maxnotimeout`.

After the `keepAlive()` message is sent from a Crawler, it will synchronise the already discovered web pages with the URL repository and `memcached`. By doing this, the Observatory has a snapshot of the pages discovered and can restart from this point if a Crawler hangs.

KeepAlive message:

Parameter	***Description***
PID	PID (Process Identifier) of the instance running as integer, e.g. 1234
Machine	Machine name the current Crawler is running on, e.g. 'eiao4.hia.no'
Returns None	

Table 8: Paramters for keepAlive from each Crawler

notAlive message:

Parameter	***Description***
Machine	Machine name the Crawler daemon is running on, e.g. 'eiao4.hia.no'
Returns list of PIDs of crawlers that have not reported keepAlive	

Table 9: Parameters of notAlive from the crawlers daemon

Figure 5.3 shows a sequence diagram describing how a Crawler is terminated, if the watchdog times out. In this example the Site URL Server and URL repository are run on a central location, while crawlers deamon and all instance of crawlers are run on eiao4.hia.no. In this example, the instance of the Crawler has PID 1234.

There is a mutex in `siteurlserver` to avoid race conditions when accessing the common datastructure that maintains watchdog information.

5.3.2 Watchdog functionality for crawlers that exceed memory usage

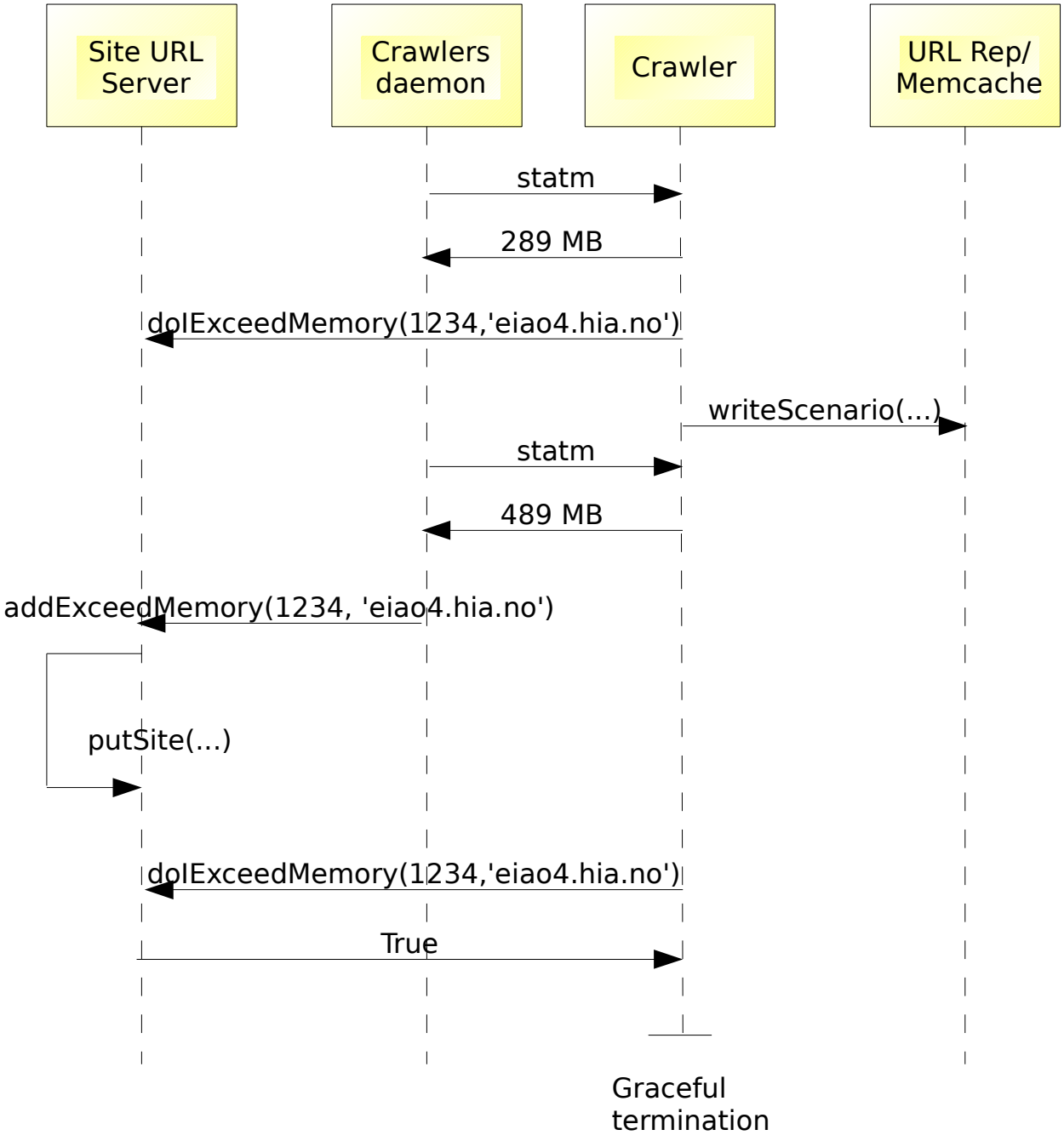


Figure 5.4: Example of the watchdog terminating a Crawler using too much memory

Similar to the implementation described in 5.3.1, a watchdog for crawlers that exceed memory usage has been implemented. The crawlers at regular intervals report the memory usage of each crawler process. Any crawler that exceeds the predefined memory limit will be reported to the Site URL server, including PID and Server information.

The crawlers daemon checks each crawler by using the memory statistics property in the proc filesystem of the process (/proc/<PID>/statm). E.g. for a crawler with PID 1234, the crawlers daemon checks /proc/1234/statm. Any Crawler that exceeds the memory usage is reported to the Site URL server with the `addExceedMemory()` message outlined in Table 10.

addExceedMemory message:

Parameter	***Description***
PID	PID (Process Identifier) of the instance running as integer, e.g. 1234
Machine	Machine name the crawlers is running on, e.g. 'eiao4.hia.no'
Returns none	

Table 10: Parameters of addExceedMemory from the crawlers daemon

Furthermore, each Crawler sends a `doIExceedMemory()` message to the site URL server, as outlined in table 11. If it returns True (the Crawler exceeds memory usage), it shuts down gracefully [29]. Furthermore, the corresponding site is pushed back on the list of sites scheduled for crawling in the site URL server.

doIExceedMemory message:

Parameter	***Description***
PID	PID (Process Identifier) of the instance running as integer, e.g. 1234
Machine	Machine name the Crawler is running on, e.g. 'eiao4.hia.no'
Returns a boolean value whether the Crawler exceeds memory or not.	

Table 11: Parameters of doIExceedMemory from the crawlers daemon

Note that this implemented change also includes continuously writing page information to the URL repository instead of adding this information at the end of a crawl. However, there can be up to a minute delay between actual download of a page and writing this information. Because of this, some URL information is expected to be lost whenever a Crawler is restarted, but this information will be quickly retrieved by the Crawler from the web site again.

Figure 5.4 shows an example of how a Crawler is terminated when it exceeds the maximum memory usage of 300 MB. In this example the Site URL Server and URL repository are run on a central location, while crawlers deamon and all instance of crawlers are run on eiao4.hia.no. In this example, the instance of Crawler has PID 1234.

29 Note that even though the Crawler has a “normal” exit, the site information is not sent to the Sampler.

5.3.3 Crawlers module

Location: robacc/Crawler

Module	*Description*
crawlers	Crawler manager.
setup.py	Installation script
init.d/crawlers	Init script for Fedora to start the crawler manager.

5.4 Crawler – EIAO branch of HarvestMan crawler

In the following chapters we present the main updates done to the HarvestMan crawler[30] in thre EIAO branch of HarvestMan (EIAOHarvestMan).

5.4.1 Crawler and fetcher thread state machines

In order to make the crawler more stable, we have implemented a state machine keeping track of the state of all crawler and fetcher threads. This state machine ensures that the crawler exit criterion is deterministic and the state handling class also includes functionality that logs and/or raises an exception if a crawler performs an illegal state transition. This has improved the stability of the crawler substantially.

Fetcher state machine

The image in Illustration 4 shows the updated HarvestMan[31] fetcher state machine and crawler end criterion.

30 The crawler is largely based on HarvestMan [HarvestMan].

31 Note that these state machines are only for EIAOHarvestMan, not HarvestMan. However, the changes done are planned to be merged back into HarvestMan.

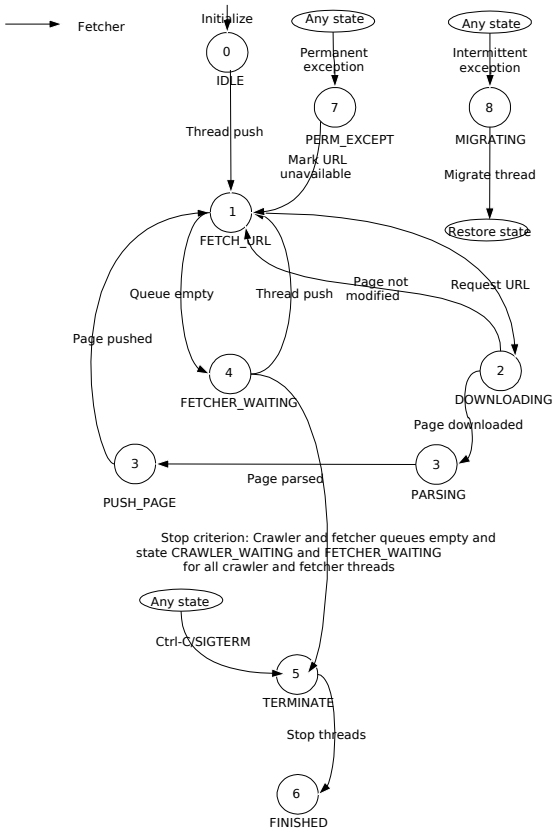


Illustration 4: Fetcher state diagram.

The fetcher is initialised in state IDLE. When a thread pushes an URL to be crawled to the URL queue, it moves to state FETCH_URL. The fetcher will then request a URL from the URL queue and move to state DOWNLOADING. If the page is not modified, which implies that more uncrawled data exists in the URL repository, then the fetcher will move back to state FETCH_URL. If the page was downloaded, then the fetcher moves to state PARSING, and parses the HTML page for links. When the parser is finished, the fetcher moves to state PUSH_PAGE, and pushes the page links to the crawler queue. When all page URLs have been pushed, the fetcher moves back to state FETCH_URL, to fetch the next URL to download. If the fetcher queue is empty, then the crawler moves to state FETCHER_WAITING.

The fetcher stop criterion is that both the crawler and fetcher queues are empty, and that all crawlers are in state CRAWLER_WAITING and all fetchers are in state FETCHER_WAITING. In this case, or if SIGTERM is received (e.g. Ctrl-C), then the crawler moves to state TERMINATE, then stops all threads and moves to state FINISHED.

Two kinds of exceptions can occur within the fetcher:

- Intermittent exceptions will cause thread migration to a new thread, the state is MIGRATING whilst migrating, and the previous state is restored after the migration.

- If a permanent exception is received, then the fetcher changes to state PERM_EXCEPT. The URL being processed is then marked as unavailable, and the fetcher moves to state FETCH_URL to get the next URL to process.

Crawler state machine

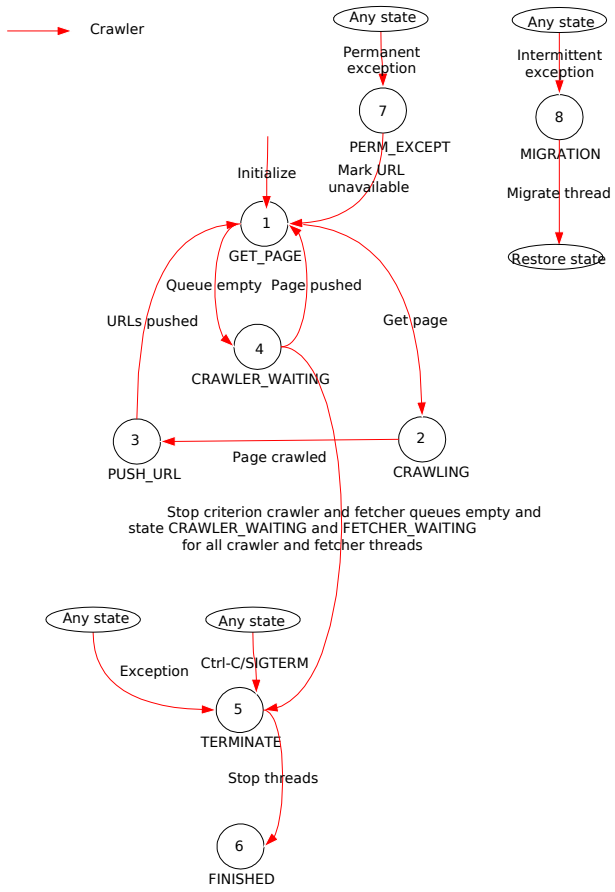


Illustration 5: Crawler fetcher state diagram.

The crawler thread is initialised in state GET_PAGE. It will then try to fetch a page from the Crawler queue, and move to state CRAWLING. The crawler will after that loop through the URLs within the page and check if the page is already crawled and check if the URL is within scope. If everything is OK, the crawler moves to state PUSH_URL. When all URLs are pushed to the URL queue, the crawler moves back to state GET_PAGE.

The crawler stop criterion is that both the crawler and fetcher queues are empty, and that all crawlers are in state CRAWLER_WAITING and all fetchers are in state FETCHER_WAITING. In this case, or if SIGTERM is received (e.g. Ctrl-C), then the crawler moves to state TERMINATE, then stops all threads and moves to state FINISHED.

Two kinds of exceptions can occur within the crawler:

- Intermittent exceptions will cause thread migration to a new thread, the state is MIGRATING whilst migrating, and the previous state is restored after the migration.

- If a permanent exception is received, then the Crawler changes to state PERM_EXCEPT, the URL being processed is marked as unavailable, and the fetcher moves to state GET_PAGE to get the next set of page URLs to process.

5.4.2 Last-modified timestamp handling

During the breadth-first scan, the web crawler will store the URL header field last-modified timestamp in the URL repository, if present. During subsequent crawls, the web crawler will query only the HTTP header and compare the last-modified timestamp to the stored value. If the web page is modified, then the entire web page will be queried in an additional HTTP GET request.[32] This limits excessive use of bandwidth for retransmission of URLs that already are up-to-date in the URL-repository. Note that the last-modified timestamp needs to be checked, to ensure that the timestamp is within valid bounds. Reasons to reject the timestamp, are:

- Timestamp = -1
- Timestamp older than 1/1-1980
- Timestamp newer than current date

The URL repository also stores information about whether a given domain supports if-modified-since. If the web server supports if-modified-since, then the crawler should prefer to use HTTP if-modified-since instead of querying the HTTP header, since this is more efficient. (It avoids the second HTTP request in the scenario above.)

5.4.3 Least recently used cache

The Observatory uses an intermediate in-memory dictionary to store identified URLs. Whenever this dictionary grows larger than 6 000 URLs identified[33], the least recently used URLs are flushed to a local disk cache using cPickle, to avoid large queues in memory. Note that this is in contrast to the list of all identified URLs that is handled with `memcached`, as described in chapter 5.4.6.

5.4.4 Crawling according to scope

At the beginning of a crawl towards a web site, the crawler extracts corresponding scoping rules from the URL repository and compiles each regular expression.

Furthermore, whenever a new URL is detected by the crawler, it then pushes this URL to the download queue only if it matches the scoping rules. In other words, the crawler will only follow links that match the scoping pattern. Note that whenever the crawler finds a match in one of the scoping rules, the subsequent rules are not applied. Because of this, the order of the scoping rules is important. A URL that does not match any of the scoping rules, will automatically be discarded.

32 Note that this results in a crawling strategy different from breadth-first.

33 The LRU cache was developed in order to manage exhaustive scans of large web sites. This strategy is effectively not used, now that we have a max limit of 6000 URLs crawled.

For reason of clarity, following are four examples. The scoping rules that apply are the same as defined in table 7 in chapter 5.2.1.

Example 1:

URL: `http://www.example.com/site1/admin/adminpage.html`

If the URL above is detected, such as linked from another page, the Crawler apply the following scoping rules; 1, 2, 3 and 4. Scoping rule 4 matches the URL and the following scoping rule (5) will not be applied. Since scoping rule 4 is defined as excluding, the URL will not be added to the download queue.

Example 2:

URL: `http://www.example.com/site1/admin/public/publicadminpage.html`

If the URL above is detected the Crawler will apply the following scoping rules; 1 and 2. Scoping rule 2 matches the URL and the following scoping rules (3, 4 and 5) will not be applied. Since scoping rule 2 is defined as including, the URL will be added to the download queue. Note that even though the URL matches the excluding scoping rule 4, it will still be included (since this rule will never be applied).

Example 3:

URL: `http://www.example.com/site1/normalpage`

If the URL above is detected, the Crawler will apply the following scoping rules; 1, 2, 3, 4 and 5. Scoping rule 5 matches the URL and all the scoping rules have been applied. Since scoping rule 5 is defined as including, the URL will be added to the download queue.

Example 4:

URL: `http://www.anotherexample.com/site1/normalpage`

If the URL above is detected the Crawler will apply the following scoping rules; 1, 2, 3, 4 and 5. None of the scoping rules matches the URL. Because of this, the URL will not be added to the download queue.

5.4.5 URL collections to support framesets

The URL collection/url context concepts in Harvestman supports framesets/frames internally. This is described in the `HarvestManFrameContext` class in the `urlcollections.py` module.

5.4.6 Crawler functionality for writing URLs to memcached and URL repository.

Whenever a `keepAlive()` message is sent to the Site URL server, the crawler sends URLs to the URL repository. Additionally, the local `downloaded` dictionary is flushed. This is to avoid making internal Crawler dictionaries grow extremely large for large sites.

The URL repository adds all new URLs to both `memcached` [MEMCACHED] and PostgreSQL. `Memcached` is a high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. The crawler uses it for fast verification of whether a URL has been visited or not, since very large Python dictionaries are too slow and use too much memory. PostgreSQL is used to store information about all URLs crawled in the URL repository.

The URLs are written to `memcached` using the following format:

{'hash of url',(url,scenarioid,domain,pageindex)}

The reason the URLs being stored as hash values, is because `memcached` does not support dictionary keys larger than 250 bytes[34]. In contrast, URLs might be a lot larger. The URLs are additionally stored in the page table of the URL repository.

5.4.7 Functionality for checking if an URL already has been downloaded

Whenever the Crawler checks if a URL already is downloaded, it first checks its local dictionary. If it is not there, it then checks `memcached`. If it is not stored there either, the check returns false. Memcached has sufficient capacity to store information about all URLs identified for all crawlers.

This works as following (pseude-code):

```
def isdownloaded(url)
 if url localmemorydictionary:
    return True
 if url in memcache:
    return True
 return False
```

Note that the local dictionary is periodically flushed to avoid large dictionaries.

The URL cache is flushed when a new crawl is loaded with the `eiaoassess` command.

34 Note that there is a very small, but real risk of hash collision between URLs. A hash collision would mean that a random URL that by chance had the same hash value as an existing URL stored in memcached. The consequences of this would be that the second URL would not be stored in the URL repository, which is not a big loss, since another URL would be stored instead. Since the MD5 hash algorithm is 128 bits, the chance of a collision for max 6 000 URLs is so small that it can be ignored for all practical purposes.

5.4.8 JavaScript handling

HarvestMan contains JavaScript handling that uses a pure Python port of Rbnarcissus [RBN]. The `narcissus.py` module contains the main parser code. The `jsparse.py` module prints out a dictionary of functions in a javascript source file passed as argument to it.

The module consists of two classes. `HTMLJSParser` is an HTML Javascript extractor which can extract javascript code present in HTML pages. `JSParser` builds upon `HTMLJSParser` to provide a Javascript parser which can parse HTML pages and process Javascript which performs DOM modifications. Currently this class can process `document.write()` functions and Javascript based redirection which changes the location of a page. Both classes are written trying to mimic the behaviour of Firefox (2.0) as closely as possible.

Now the parser closely approximates the original ruby parser. The parser passes for all sample files directly inside the "js" folder. The ruby parser also exhibits the same behavior.

The main limitations:
- It is currently is not a full-fledged JavaScript DOM parser. Currently it parses basic `document.write(...)` without any variables, just pure strings and takes care of all forms of DOM location changes.
- The Crawler only parses inline JavaScript, not the imported JavaScript files.
- It is relatively slow, so we do not want to use it for general large-scale operations.

Experiments have shown that JavaScript handling quite seldom limits the crawls. In the 3 000 available sites available to EIAO, few web sites have so far been identified that we could not extract any links from because of Javascript. Note that web sites might have both links in a-attributes and JavaScript links. It has therefore not been put much effort into optimising the JavaScript handling.

5.4.9 Crawler modules

Locations: robacc/Crawler

Module	***Description***
crawler.py	Class for crawling a web page
crawlerwrapper	Script for starting the crawler and communicating with site URL server
crawlererror.py	Exception classes
config.template	HarvestMan configuration template
config.py	Reads HarvestMan configuration file
configparser.py	Parsing HarvestMan configuration Module
connector.py	Keeping connections to web servers
datamgr.py	Data manager module

Module	***Description***
debugcrawler	Script for starting the crawler under the WinPDB debugger
EIAOHarvestManError.py	Reporting exceptions from EIAOHarvestMan
EIAOStaticConfig.py	Static variables used in EIAOHarvestMan
harvestman.py	Main harvestman module (not used - EIAO uses crawlerwrapper instead.)
harvestmanklass.py	Main HarvestMan class
hooks.py	Module for connecting plugins to functions
install.py	Installation file
__init__.py	Package definition
logger.py	Performing logging
options.py	Commend line options
pageparser.py	Parsing (x)HTML/CSS documents
robotparser.py	Parsing robots.txt
rules.py	Crawling rules, e.g. rules for following robots.txt
urlcollection.py	Collection of URLs (e.g. frames)
urlparser.py	Parsing of URLs
urlproc.py	Escaping of special URLs
urlqueue.py	Queue of URLs to be crawled, downloaded
urlthread.py	Thread for downloading and thread for managing the download threads
urltypes.py	Types of URLs, e.g. html, image
utils.py	Utilities such as printing of help
common/common.py	Internal data structures
common/keepalive.py	Module for managing persistent HTTP connections
common/lrucache.py	Least Recently Used Cache
common/methodwrapper.py	Managing pre-post funtion calls for plugins
common/optionparser.py	Generic parser module
common/progress.py	Printing progress
common/singleton.py	Singleton module
common/__init__.py	Package definition
dev/feedparser.py	RSS parser
js/jsdom.py	Defined JavaScript DOM objects

Module	*Description*
js/jsparse.py	Print parsed functions (used by unit tests)
js/jsparser.py	Extracts JavaScript from (x)HTML
js/narcissus.py	Python port of Rbnarcissus
plugins/eiao.py	Loads plugins
plugins/eiaodatamgr.py	Checks whether a URL has been downloaded or not via memcached
plugins/*	Other plugins than the two mentioned are not used

5.5 SamplingServer - Sampling server

The sampling server is responsible for managing samplers, where each sampler is responsible for acquiring a random uniform sample of web pages from the URL repository for one web site. This server receives sites to sample from running crawlers and sends sites that have been sampled to the ETL Server.

5.5.1 Sampling server modules

Location: robacc/SamplingServer

Module	*Description*
samplingserver.py	Main module
samplingtestclient.py	Client to test the sampling server
setup.py	Installation
__init__.py	Package definition
init.d/samplingserver	Init start/stop script for sampling server

5.5.2 SamplingAlgorithm - Sampler

The main functionality of the sampler is to perform a random uniform sampling of each site based on URLs extracted by the crawler. This also includes interacting with the WAMs and store WAM results in a RDF database. An outline of the Sampler can be seen in Figure 5.5. The Sampler select up to 600 random URLs from the URL repository. Each page is downloaded and further sent to a WAM for evaluation. Note that both downloading and WAM interaction are run in separate threads, in total 10 threads of each.
Each WAM interaction thread selects which WAM to interact with at random to balance the load between the WAMs[35]. Note also that any site with less than 30 pages sampled is

35 This simple approach does not balance the load so well if the WAM servers have different capacity. A better approach might be investigated in the future.

discarded, unless the web site has been manually verified to contain less than 30 pages and the field smallsite is set to True in the site table[36].

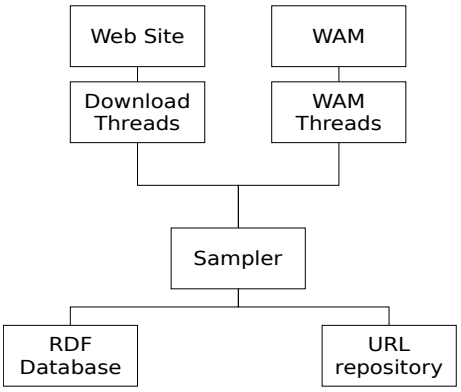


Illustration 6: Overview over sampler.

Frame context handling in Sampler

The Crawler keeps track of Frame contexts, whereas CSS downloading and cascading calculations for each (x)HTML element is performed autonomously in the WAM. The WAM is not aware of Frame contexts, and will evaluate and return EARL for each HTML resource it is sent. The Sampler will append the EARL evaluation results from all (x)HTML resources that belong to the same frame set to the same RDF scenario, so that the Data Warehouse knows that these resources belong to the same web page and should be presented together.

5.5.3 SamplingAlgorithm modules

Location: robacc/SamplingAlgorithm

Module	***Description***
Sampler	Main module
samplingerror.py	Generate sampling exceptions
setup.py	Installation description
test.py	Testing
__init__.py	Package definition

36 Early review comments indicate that it may be sufficient to require minimum 30 sampled results, and not 30 pages. This may be implemented in future versions.

5.5.4 AdaptiveSampling - CWAM, standard deviation and error margin calculations

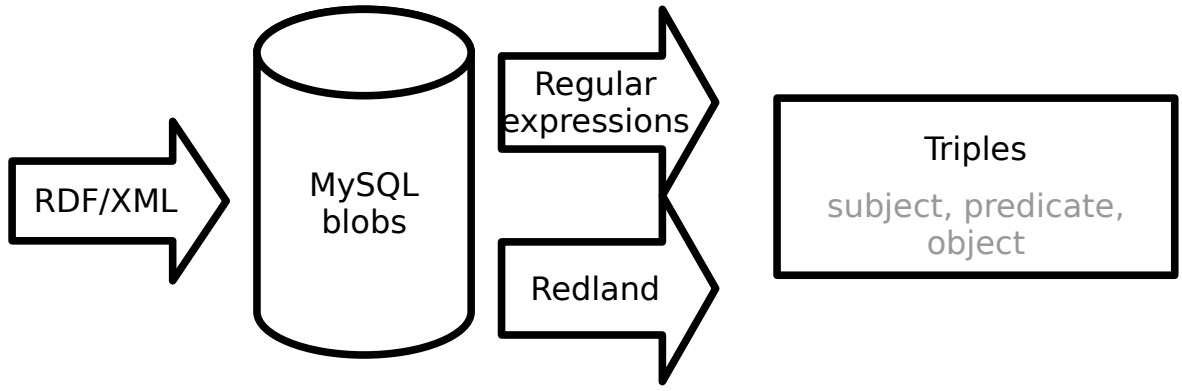
The adaptivesampling module was previously used as stop criterion for sampling. Adaptive sampling is now deprecated due to statistical issues, and may be removed in future versions of the Observatory. The pre-calculation of CWAM value, standard deviation and error margin on site level is still being used to offload the ETL and datawarehouse from these calculations.

Location: robacc/AdaptiveSampling

Module	***Description***
adaptivesampling.py	Main module that calculates average CWAM values on web site level, standard deviation and error margin for the web site being sampled. The module also defines the stop criterion for sampling to a given error margin (deprecated due to statistical issues).
adaptivesamplingerror.py	Exceptions for adaptivesampling.py
__init__.py	Package definition
test.py	Unit tests
setup.py	Installation script.

5.5.5 PyTripleStore - Python based triplestore

PyTripleStore is a highly specialized RDF-database optimised for speed for the Observatory. It uses a combination of MySQL blobs, regular expressions and Redland [Redland]. An outline of the the work flow of PyTripleStore can be seen in Drawing 1.



Drawing 1: Outline of PyTripleStore

The EARL retrieved from the WAMs, is connected to a larger RDF-graph as outlined below.

(1) TestRun
- (1) SiteSurvey
  - Barrier Indicator (Site result)
  - Variance
  - Error Margin
  - (30-600) PageScenario (page)
    - (1-*) PageSurvey (Resource)
      - HTTP Header
      - Language
      - Doctype
      - Authoring Tool
      - **EARL**

Technically, the RDF graph supports storing results from several sites. However, for performance reasons, each RDF graph contains results from only one site. Note that in addition to the EARL and surrounding RDF, some pre-calculated values are stored in the graph reducing the computational requirements of the Data Warehouse at a later stage. For instance, calculating the site results for all evaluated sites is rather computational intensive for the Data Warehouse. The computational load is lower for the samplers, which in addition are parallelised on site level. The ETL and Data Warehouse therefore utilise these results rather than calculating performing the calculations themselves.

The RDF schema with OWL restrictions for the EIAO RDF graph can be found at http://svn.eiao.net/robacc/RDFSchema/RDFSchema.rdf

Writing RDF

The resulting RDF/XML consisting of both EARL received from the WAM and corresponding RDF in the crawler, are sent to a data base pool with several threads. Each thread has a connection to a RDF database and stores the RDF/XML as MySQL blobs. Note that even though each “bulk” of RDF is stored in a separate blob, all parts of the RDF are connected together in a larger graph as outlined above. Note also than with this approach only minimum parsing of the RDF/XML is required. I.e. no DOM tree from the XML is created, in contrast to traditional approaches when dealing with XML and RDF/XML. In contrast, the calculated results, such as site results, are computed based from counting PASS/FAIL results in the EARL using regular expressions, which require minimal computational resources compared to building complete DOM trees[37].

37 Note that the regular expressions are created only for the EARL format used by EIAO. It will not be able to parse arbitrary EARL/RDF in general. This is a design decision that was taken to support as high throughput as possible of evaluation results through the observatory for large scale evaluations.

Reading RDF

The RDF database is used as input to ETL and is thus essential for the **E**xtraction-part of the ETL. However, only the RDF graph, not the underlying storage of the RDF is visible to the ETL. The mapping from the underlying storage to the RDF graph is done by a specialised python module hereby referred to as RDFreader. The ETL queries the RDFreader by a sets of triples (subject, predicate, object). Any missing information are seen as wild cards in these queries.

For reasons of understanding, we provide an example of of retrieving the Site Survey ID from the RDF graph. In this example we assume that the Site Survey ID is stored as the triple subject='200802',predicate='sitesurveyid',object='12345'.
Note also that for reasons of clarity, the example is simplified:

```
#Getting sitesurveyid
testrunid = '200802'
sitesurveytriples = RDFreader(
            subject = testrunid,
            predicate = 'sitesurvey',
            object = None)
sitesurveyid = sitesurveytriples[0]['object']
```

In the above example the the ETL queries for all triples where subject=testrunid predicate='sitesurvey'.

The RDFreader returns a list of triples that matches this query. In this case, only one triple will match and the ETL extracts the site survey ID, which in this case is stored in the first (and only) triple as an object. Note again that the underlying RDF/XML is not visible for the ETL, only the graph as a set of triples.

The RDFreader has two main methods of transforming the stored RDF/XML:

(1) **Building a DOM tree.**
For smaller parts of the graph, such as site survey information, complete DOM trees are created using Redland bindings for Python. Since these parts of the graph are small, only minimal computational resources are needed for this. All triples from the graph are stored as lists in dictionaries (see below).
(2) **Building regular expressions.**
For the larger parts of the graph, such as EARL, regular expressions are used to extract the needed triples. Note that in contrast to building DOM trees, expert knowledge of the RDF is needed. Based on the regular expressions, all triples are extracted and stored as lists in dictionaries (see below).

For fast lookup, four dictionaries are created using with the following keys;
`(subject, predicate)`,
`(subject, object)`,
`(predicate, object)` and
`(subject)`.

The dictionary with keys `(subject,predicate)` will for example contain all available triples indexed by (`subject,predicate`).

These are all populated in both (1) and (2) above, initiated by first query from the ETL. Based on this, every query from the ETL to the RDFreader is reduced to a quick in-memory dictionary lookup.

RDF page scenarios to represent web pages

When crawling, the Crawler stores information about page contexts (i.e. which web pages that belong together in a frameset) in the URL repository. Each web page in the page context will be sent to the WAM for evaluation, and the result needs to be stored in the RDF database that is sent to the ETL server for loading into the Data Warehouse. The EIAO RDF page scenario structure is used to store information about which parts of the web page that belong together in the RDF database.

For example, if one assumes that file.html refers a frameset consisting of two HTML files a.html and b.html, and that all files are evaluated by the EIAO WAM. The RDF page scenario would then consist of the EARL result from each of these HTML files, as indicated in the RDF example below:

```
<eiao:pageSurvey rdf:about=”PageSurvey_39392”>
  <eiao:downloaded>2006-10-16T14:37:31.277877</eiao:downloaded>
  ...
  <eiao:rangeLocation rdf:about=”PageSurvey_39392_range_1”/>
  <eiao:rangeLocation rdf:about=”PageSurvey_39392_range_2”/>
  <eiao:rangeLocation rdf:about=”PageSurvey_39392_range_3”/>
</eiao:pageSurvey>

<eiao:rangeLocation rdf:about=”PageSurvey_39392_range_1”>
    ...
   <eiao:page>http://example.com/file.html</eiao:page>
   <earl:asserts rdf:resource=”someassertions_1”>
   ...
   <earl:TestSubject  rdf:resource=”somesubject_1” />
   ...
</eiao:rangeLocation>


<eiao:rangeLocation rdf:about=”PageSurvey_39392_range_2”>
    ...
   <eiao:page>http://example.com/a.html</eiao:page>
   <earl:asserts rdf:resource=”someassertions_223”>
   ...
   <earl:TestSubject  rdf:resource=”somesubject_2” />
   ...
</eiao:rangeLocation>


<eiao:rangeLocation rdf:about=”PageSurvey_39392_range_3”>
    ...
   <eiao:page>http://example.com/b.html</eiao:page>
   <earl:asserts rdf:resource=”someassertions_234”>
   ...
   <earl:TestSubject  rdf:resource=”somesubject_3” />
   ...
</eiao:rangeLocation>
```

5.5.6 PyTripleStore modules

Location: robacc/PyTripleStore

Module	***Description***
DBPool.py	Pool of RDF database writer threads
RDFGenerator.py	Generate RDF/XML from RDF model
RDFGeneratorerror.py	Exceptions for the RDF generator
RDFreaderwriter.py	Writing and Reading RDF to/from database
RDFreader2.py	Read and parse RDF blobs from database
asynchRDFWriter.py	Asynchronous RDF writer
deletetriple.py	Deleting a triple in the RDF database
pytriplestore.py	Main Module
setup.py	Installation module
tables.sql	Data definition for RDF database
test.py	Test module
earlexample.rdf	Example data used by test module
__init__.py	Package definition

5.6 WAMs - WAM server

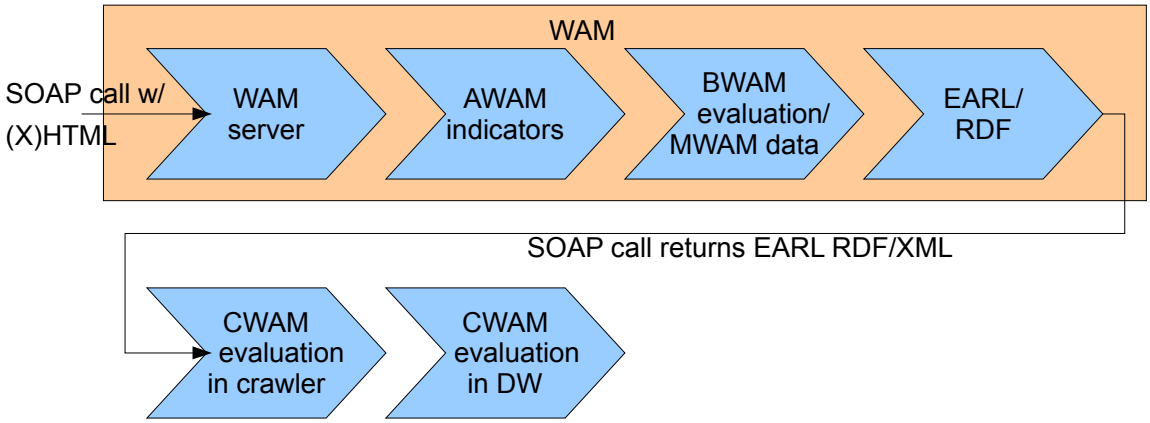


Illustration 7: WAM production line

The WAM server typically consists of a SOAP server that provides one function `check()` that can be used to perform accessibility evaluations. The `check()` function will first extract AWAM indicators from the document. AWAM indicators can be extracted in different ways - the WAM does not care how they are produced. EIAO does for example use Schematron to extract AWAM indicators from (X)HTML documents, and we use separate AWAM test classes to extract AWAM indicators from CSS documents and to perform CSS and HTML validation. All AWAM modules must however produce a valid AWAM result structure, which consists of a dictionary indexed by AWAM id that has a dictionary indexed by location and with AWAM result as value. There can be several locations per AWAM ID, but the locations must be unique per AWAM ID. In Python the AWAM result data structure can be represented as:

```
resultMap = {"EIAO.A.10.7.4.2.HTML.2.1":{(1,10):url1, (1,20):url2},
             "EIAO.A.10.4.3.1.HTTP.1.1":{(0,0):contentLanguage}}
```

This means that one can index the AWAM data structure using the AWAM id and the location, represented as a tuple:

```
resultMap["EIAO.A.10.7.4.2.HTML.2.1"][(1,10)]==url1
```

The BWAM and MWAM instances operate on the AWAM result data. The B and MWAM rules classes are instantiated and bound to the AWAM data with the method:

```
rules=AbstractWAM.WamRules(resultMap,ruleset)
```

All BWAM rule instances that have locations with AWAM results will calculate a statement about accessibility based on one or more AWAM indicators.

In a similar fashion will MWAM instances that have locations with AWAM results produce content statistics.

The WAM server will typically perform the WAM tests by iterating through all BWAM instances, and iterate through all BWAM results that each BWAM test generates. EARL assertions are created with the function `report.assertion(result)` if the result is from a BWAM and `report.metaData(result)` if the result is content statistics from an MWAM.

The EARL report is then assembled with `report.earl()`. This EARL report is returned from the `check()` SOAP function to the Sampler that called the WAM. The file `WAMs/relaxed_wam/wam_container.py` can be used as design pattern for the BWAM test and EARL generation part of a WAM.

5.6.1 WAM call interface

The WAM server binds by default to port 8888, and it publishes one SOAP method `check()`, that returns EARL.

```
earl=soapserver.check(url,cont=None,contentType='text/html',
                      encoding='utf-8',ruleset=None,httpheader={},
                      compression=None)
```

Arguments:

Argument	Required	Description
url	mandatory	URL of document to be evaluated.
cont	mandatory	Content of document evaluated.
contentType	mandatory	Content type, default 'text/html'.
encoding	mandatory	Encoding of document content, default is 'utf-8'.
ruleset	optional	Which set of BWAM rules to use. Default is all.
httpheader	mandatory	HTTP header
compression	optional	May be used if a compression scheme is supported by the WAM. Currently the relaxed WAM only supports compression=”base64” to avoid some instability problems in SOAPpy.

Table 12: Arguments of WAM servers check() method.

Typical use of the WAM call interface is shown below:

```
import SOAPpy
import base64
import urllib2
# Download web page, find character set and content type.
f = urllib2.urlopen(url)
html=f.read()
ctype=f.headers["content-type"].split(";")[0]
charset=f.headers["content-type"].split("=")[-1]
# Connect to WAM SOAP server
server = SOAPpy.SOAPProxy("http://localhost:8888")
# Base64 encode content
content=base64.encodestring(html)
# Send a check request to the WAM
earl = server.check(f.geturl(),content,ctype,charset,"all",
                    httpheader=dict(f.headers),
                    compression="base64")
```

More detailed examples can be found under `robacc/WAMs/relaxed_wam/tests`.

5.6.2 WAMs/wamlib - WAM library

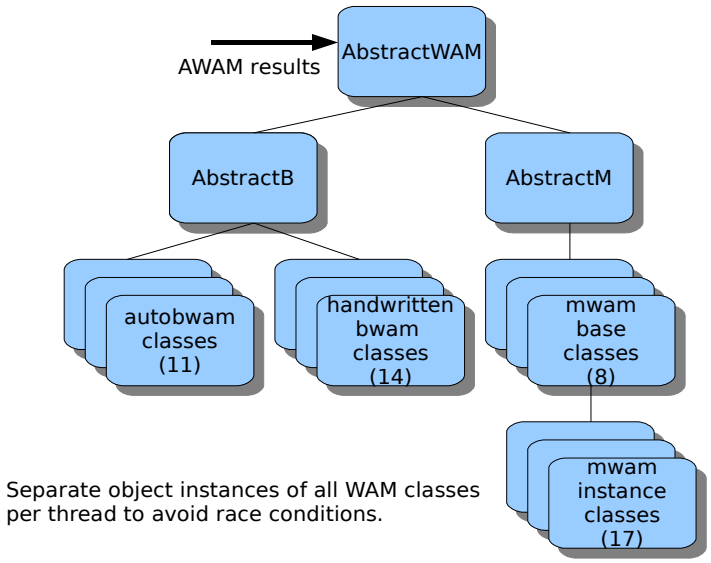


Illustration 8: WAM class hierarchy.

B-WAMs compile statements about accessibility based on AWAM indicators. MWAMs return content statistics data based on AWAM indicators.

The `AbstractWAM` class is the base class that both B- and MWAMs are derived from. The `AbstractB` and `AbstractM` WAM differ only in the parameters sent in to the constructor of `AbstractWAM`. The `AbstractB` has type `AbstractB.BWAM`, whereas the `AbstractM` has a different WAM ID and type that repends on type of content statistics, as described in `WAM_Results.py`. To support the set of BWAMs in D3.3.2, 11 classes have been automatically generated from the D3.3.2 specification in the module `autobwam.py` and 14 bwam classes have been written manually in `bwam.py`. The MWAMs for content statistics consist of 8 (abstract) base classes and 17 instance classes generated from the base classes.

The module `wam_container.py` is a ready-made BWAM module that can produce EARL based on the UWEM 1.2 fully automatable test set [UWEM]. It is intended to be generic and can be extended with new tests, if needed.

5.6.3 Default return value for WAMs.

It is possible to configure the B- or MWAM to return a default value in case no AWAM results exist by setting the parameter `default=DEFAULT_PASS` or `DEFAULT_FAIL` in the `AbstractB` or `AbstractM` constructor respectively. For example:

```
class UWEM_B_10_1_1_3_HTML_DEF_6_1(AbstractB):
   def __init__(self,awamresult):
      AbstractB.__init__(self,awamresult,"UWEM.B.10.1.1.3.HTML.DEF.6.1",
          ['EIAO.A.10.1.1.2.HTML.2.1'],default=DEFAULT_PASS)
      self.title = 'The <embed> element is used.'
      self.description = """Checkpoint 1.1: ..."""

   def result(self,s):
      return  int(self.aWAM("EIAO.A.10.1.1.2.HTML.2.1",s))

AppendKlass(UWEM_B_10_1_1_3_HTML_DEF_6_1)
```

This example adds a new AWAM class for the test UWEM.B.10.1.1.3.HTML.DEF.6.1 that has a default value of PASS if no results exist in the AWAM EIAO.A.10.1.1.2.HTML.2.1 for location s. The location s contains the current AWAM result handled by the AbstractWAM iterator for the AWAM IDs referred to in the constructor of the BWAM (here EIAO.A.10.1.1.2.HTML.2.1).

Note that without the `default` parameter, the WAM will not return any results unless the AWAM Ids used by the WAM contain locations with results. The method `self.aWAM(awamID,s)` is a convenience function that does exception handling for indexing into the AWAM data structure. It returns 0 (False) in case a referenced AWAM ID is not present, which can happen for BWAMs that refer to several AWAM Ids.

5.6.4 WAM library modules

Location: robacc/WAMs/wamlib

Module	*Module Description*
abstractWAM.py	Abstract class for WAM tests. It also defines the abstract subclasses AbstractB for BWAMs and AbstractM for content statistics MWAMs. Every BWAM class needs to inherit from AbstractB, and every MWAM class needs to inherit from AbstractM.
autobwam.py	Automatically generated BWAMs from D3.3.2 by the bwamparser in `WAMs/wamlib/bwamparser`.
bwam.py	Manually written BWAM classes (the ones that could not be automatically generated.)
mwam.py	Content statistics MWAMs.
WAM_Results.py	WAM_Results.py contains BWAM and MWAM result objects and utility methods and constants used for generating EARL.
earl.py	Module for EARL generation. It takes BWAM_Result objects from the BWAM assessments, and wraps them in to EARL/XML.
version.py	Version information.
wam_container.py	WAM container module, which performs BWAM tests on AWAM results and produces EARL.
wamerror.py	WAM exception classes.
wamlogger.py	WAM logging functions.
StopIteration.py	Dummy exception class to be able to use iterators with Jython.
mimetypes.py	Python mimetypes library patched with some missing mime types.
timeprofile.py	Time profiling module.
setup.py	Installation script for wamlib.
__init__.py	Package definition
bwamparser/*	Parser to generate autobwam.py from WAM specification.

5.6.5 WAMs/relaxed_wam - EIAO Relaxed based WAM.

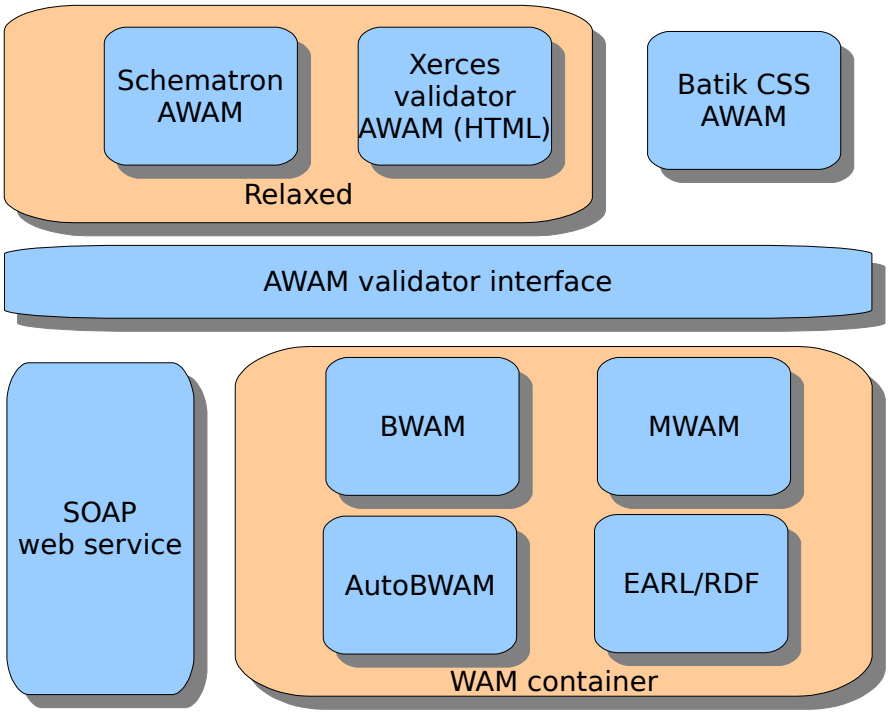


Illustration 9: WAM architecture for EIAO Relaxed based WAM.

The EIAO WAM server supports the UWEM 1.2 test set, as specified in D3.3.2 WAM specification. The largest technical change from the previous version of the Observatory, is that CSS handling now supports cascading calculations using the Apache Batik CSS library.

The architecture of the EIAO Relaxed based WAM is shown in illustration 9. The SOAP web service interface and the WAM container have been described in the previous sections. This section describes how the WAM server works. Its source code can be found in `robacc/WAMs/relaxed_wam/WamServer.py`. The SOAP web service module contains the `check()` function that first calls the AWAM modules, and then the BWAMs in the WAM container (`wam_container.py`).

In illustration 9, the SOAP web service is split out from the WAM container, since it it useful to have the flexibility to be able to exchange the SOAP web service based middleware with other types of middleware in the future, if needed.

The EIAO WAM starts a multithreaded WAM server on port 8888 and registers the `check()` method of the `WSWAMServer` class in the file `WamServer.py`. It then starts running the SOAP server with `server.serve_forever()`. Whenever the WAM gets a `check()` request, to perform an accessibility evaluation, it will first check if the content was base64 encoded. If it was, then it will be decoded. The WAM server will then create an

instance of the Relaxed validator with `a=AwamValidatorPy.AwamValidator()` and it will then perform AWAM validation using the EIAO Schematron AWAM rules in Relaxed and perform HTML validation using Xerces by calling the method `a.ValidateAWAM()`, as indicated below:

```
resultMap=a.ValidateAWAM(resultMap,url,cont,contentType,encoding)
```

The data structure `resultMap` contains the AWAM result data structure.

The WAM will then create an instance of the Batik CSS WAM and depending on content type, the CSS WAM will perform case insensitive evaluation (HTML) or case sensitive evaluation (XHTML). The CSS WAM is typically called like this (plain HTML example):

```
cssWam=CSSWam.CSSWam(resultMap,type=CSSWam.HTML,
                     media=mediatype,xmlBase=url,
                     httpHeader=httpheader)
cssWam.feed(cont)
```

The CSS WAM will add AWAM results to the `resultMap` AWAM data structure for tests that are applicable. When all AWAM indicators from the HTTP header also have been set, the WAM server will perform the BWAM tests in the WAM container on the AWAM results, and return EARL, which is returned from the `check()` method of the WAM server, as shown below:

```
return wam_container.bwams(url,resultMap,ruleset)
```

5.6.6 EIAO Relaxed WAM modules.

Location: robacc/WAMs/relaxed_wam

Module	*Module Description*
WamServer.py	The multithreaded WAM server that defines the SOAP check() method, performs AWAM evaluations BWAM evaluations and returns EARL.
AwamValidatorPy.py	Jython interface to the Java based Relaxed HTML validator with EIAO modifications.
jythonRunner.py	Helper module to start Jython from Cpython with all required arguments.
relaxedwam	The main script that starts WamServer.py using Jython. It also reads required settings from the EIAO system configuration file /etc/eiao/initial.rdf.
init.d/relaxedwam	Init script to start/stop the relaxedwam service.
version.py	WAM version information.

Module	*Module Description*
sc2.py	Simple parser to parse the /etc/eiao/initial.rdf system configuration file, to avoid dependencies to Redland.
SOAPpy-0.12.0/*	Jython patched version of SOAPpy (Python SOAP library).
tests/batch-dispatcher.py	Test WAM with a list of URLs as argument.
tests/dispatcher.py	Test WAM with one URL as argument.
tests/file-dispatcher.py	Test WAM with a HTML file as argument.
tests/test.py	WAM unit tests.
tests/__init__.py	Package definition.
fpconst.py	Needed by SOAPpy to handle IEEE 754 floating point values.
httplib.py	Patched version of httplib for Jython.
__init__.py	Package definition.
setup.py	Installation script.

Implementation of AWAM modules used by the EIAO WAM is shown in the following sections.

5.6.7 WAMs/CSSWam - CSS A-WAM

The AWAM CSS handling follows the CSS cascading rules and supports attachment to HTML elements, so that only CSS rules that are applied in the on-line reports are tested. Apache Batik has been chosen as design base, since it is a mature Java library that supports CSS and cascading. The CSS WAM handling is specified in D3.3.2 WAM specification.

Mapping HTML based CSS handling to SVG/XML CSS

Batik is an SVG/XML based CSS library so that a mapping from HTML based CSS handling to SVG/XML CSS handling is needed by the Observatory. More information about associating stylesheets with XML documents, can be found in the W3C XML stylesheet recommendation[XMLCSS].

Mapping HTML style attribute to SVG/XML.

No mapping is needed, since SVG/XML supports the style attribute[38].

Mapping HTML <style> element to SVG/XML.

No mapping is needed, since SVG/XML supports the <style> element.

38 Note that Apache Batik needs to be patched to make non-SVG elements stylable.

Mapping HTML <link> elements to SVG/XML.

Style Sheets can be associated with an XML document by using a processing instruction whose target is `xml-stylesheet`. This processing instruction follows the behaviour of the HTML 4.0 `<LINK REL="stylesheet">`.

The `xml-stylesheet` processing instruction is parsed in the same way as a start-tag, with the exception that entities other than predefined entities must not be referenced.

Example from [XMLCSS]:

```
<LINK href="mystyle.css" rel="style sheet" type="text/css">
<?xml-stylesheet href="mystyle.css" type="text/css"?>

<LINK href="mystyle.css" title="Compact" rel="stylesheet"
type="text/css">
<?xml-stylesheet href="mystyle.css" title="Compact" type="text/css"?>

<LINK href="mystyle.css" title="Medium" rel="alternate stylesheet"
type="text/css">
<?xml-stylesheet alternate="yes" href="mystyle.css" title="Medium"
type="text/css"?>
```

CSS cache

EIAO has added CSS caching functionality to Batik, to support re-using CSS stylesheets whenever possible, instead of downloading and reparsing the CSS stylesheet for every web page evaluated. Here is the complete documentation on the new APIs introduced:

`enableCaching(boolean)` - Turns caching on by setting a flag. If false is passed, caching is turned off.

`createCache(size)` - Creates the cache (allocates memory for the hash table), with the given initial size. The hash table has a default load factor of 0.75 . This also enables caching automatically.

`clearCSSCache()` - Clears the cache and removes all keys. Effectively the engine will re-download and reparse stylesheets for all cached URLs.

`clearCSSCacheForURL(url)` - Removes the cache entry only for the given URL, if found.

`getCache()` - Returns the cache object (hash table).

`setCache(cache)` - Sets the cache object to 'cache'. This automatically enables caching.

5.6.8 CSS AWAM modules

Location: robacc/WAMs/CSSWam

Module	*Module Description*
CSSWam.py	Batik based CSS AWAM.
csswam	Script to test and debug CSS AWAM. Use this script to run the unit tests.
eiaoBatik.patch	Patch to Apache Batik to support the CSS AWAM.
jythonRunner.py	Helper module to start Jython from Cpython with all required arguments.
jythonconsole	Helper program to start Jython to debug the CSS AWAM and run unit tests.
test.py	Unit tests.

5.6.9 RelaxedWAM - Relaxed HTML validator based AWAM.

The Relaxed HTML validator[RELAX] is used as basis for the EIAO Schematron[SCHEM] based AWAM. EIAO has adapted a set of UWEM 1.2 compliant Schematron rules that implement most of the AWAM tests. Relaxed uses 3 XSLT transformations to get the final validation result. The compiled XSLT stylesheets are cached during the Schematron processing to improve the WAM evaluation speed.

HTML Tidy Fallback handling in WAM

The HTML Tidy fallback handling for accessibility evaluations is used to perform graceful degradation for web sites where Relaxed fails to parse the document. Common reasons for this, is documents marked with the wrong document type (e.g. marked as XHTML but in reality plain HTML.) The procedures below sums up the handling in the WAM.

1. To enable tidy set the new project property "html.errors" to "cleanup" in project.properties file of Relaxed. If set to anything else, tidy will not be invoked and the behavior reverts to the state before this bug-fix.

2. When a cleanup is performed with tidy, Relaxed still reports the original DOCTYPE. If there is no reported doctype, then an empty string is returned.

3. The fix also has logic to retry the page again with the default settings (no tidy) in case tidy failed in cleaning up the page so that the original HTML error is reported.

The ID for the new AWAM indicating whether tidy has been used to clean up the HTML is:

EIAO.A.0.0.0.2.HTML.12.1

5.6.10 EIAO specific files or modified files in Relaxed

Location: robacc/RelaxedWAM/relaxed

Directory/ file	***Description***
relaxed/ant setup.py	Installation script that calls Ant to rebuild Relaxed. In addition, build.properties and build.xml have been modified.
relaxed/devlib	Some new jar archives have been added. The most important are Tidy.jar for HTML tidy, myxalan.jar for schematron processing with line number information and log4j for logging to syslog.
relaxed/conf eiao-public-id-mapping.xml	Relaxed XML mapping file customised for EIAO, mapping HTML/XHTML doctypes to wcag rng rules files. The script switcher.py can be used to switch between different versions of the EIAO public id mapping files. The latest version, R3, is UWEM 1.2 compliant and should be used.
relaxed/conf/transformation eiao-schematron-message-saxon.xsl	Relaxed schematron style sheet for Saxon, customised for EIAO (Deprecated, since Saxon does not support reporting the line and column number of evaluation results.)
relaxed/conf/transformation eiao-schematron-message-xalan.xsl	Schematron style sheet for Xalan-J, added by EIAO.
relaxed/conf/transformation eiao-skeleton1-5.xsl	Skeleton Module for the Schematron 1.5 XML Schema Language, customised for EIAO.
relaxed/conf/transformation line-number-addon-xalan.xsl	Line/Column number addon XSL style sheet for Xalan-J parser in EIAO Relaxed.
relaxed/conf/schema/rng xhtml-frameset-eiao-R3-wcag.rng	EIAO rng schema for xhtml doctype *"-//W3C//DTD XHTML 1.0 Frameset//EN"*.
relaxed/conf/schema/rng xhtml-transitional-eiao-R3-wcag.rng	EIAO rng schema for xhtml doctype *"-//W3C//DTD XHTML 1.0 Transitional//EN"*.
relaxed/conf/schema/rng xhtml-strict-eiao-R3-wcag.rng	EIAO rng schema for xhtml doctype *"-//W3C//DTD HTML 4.0//EN"*.

Directory/ file	***Description***
relaxed/conf/schema/rng/modules eiao-R3-wcag.rng	Schematron rules for HTML AWAM as specified in D3.3.2.
relaxed/lib relaxed.jar, batik-all.jar,...	Relaxed jar file used by EIAO WAM server. In addition, the Batik jars are stored here.
relaxed/src/edu/petrnalevka/relaxed/ SchematronXalanTransformer.java DTDValidator.java	SchematronXalanTransformer.java Schematron transformer for Xalan-J XSLT parser for EIAO Relaxed. DTDValidator.java does the HTML Tidy handling.
relaxed/src/edu/petrnalevka/relaxed/error XalanErrorHandler.java, DTDErrorHandler.java, XalanCSSErrorHandler.java, EIAO_A_10_3_2_2_CSS_1_1.java	Error/message handlers. XalanErrorHandler.java - Xalan error handler for the Xalan based Schematron handling, DTDErrorHandler.java DTD validation errors.
relaxed/src/edu/petrnalevka/relaxed/util NullEntityResolver.java	DOCTYPE Entity resolver customised for EIAO.

5.7 ETLServer - ETL server

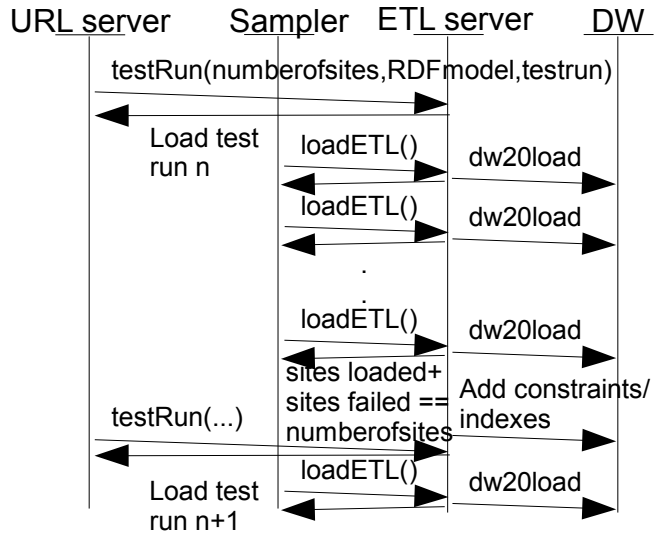


Figure 5.5: Sequence diagram showing incremental ETL load process.

The ETL starts loading data for a new test run, when it gets a `testRun(numberofsites, test run)` call. `Numberofsites` is the total number of sites to evaluate in this test run, and `test run` is the unique test run ID to load.

The Sampler stores the RDF results in a separate RDF database per web site. When a web site has been evaluated, the Sampler sends a `loadETL(RDFmodel, test run, site, dbname)` message to the ETL server, consisting of the site tested, database name needed to load the data and test run it belongs to, given that the results meet the minimum number of pages requirement.

Database credentials for the databases generated are fetched from `/etc/eiao/initial.rdf` by the `SystemConfiguration` module. The ETL load service will store ETL load requests in a queue.

The incremental load thread in the ETL service will accept `loadETL()` requests for one RDF database to load. The site results are then scheduled for loading to the Data Warehouse, and will be sent to the ETL when it is available.

Two counters keep track of number of sites loaded, and number of sites that failed loading. The load of a test run is finished when number of sites loaded + number of sites failed is the same as the numberofsites parameter sent in the initial `testRun()` message. When the test run is finished, the ETL server will finalise the load by adding Data Warehouse constraints and indexes.

5.7.1 ETL server modules

Location: robacc/ETLServer

Module	*Description*
etlserver	ETL server
init.d/etlserver	Init script to start/stop ETL service
etlservererror.py	Generate exceptions for the ETL Server
etlclient	Test program to test ETL server
setup.py	Installation procedures
__init__.py	Package description

5.8 Datawarehouse - Data Warehouse

The Data Warehouse is described in detail in D6.1.1.1 Data Warehouse functional specification.

5.8.1 Datawarehouse/dw20load – ETL

The loading of data will be performed by a 3 stage pipeline, where the first stage queries the RDF database, the second stage loads the RDF cache into RAM for fast access to the RDF, and the third stage is the incremental ETL load, which loads data from this site into the Data Warehouse. The first and second part of the pipeline corresponds to the Extract part of ETL, whereas the last part is the Transform and Load parts of ETL. The first two stages of the pipeline is implemented using two threads.

The ETL is described in detail in more detail in D6.1.1.1 Data Warehouse functional specification. For details on how the RDF graph is read by the ETL see chapter 5.5.5.

5.9 UserInterface - On-line reports

The on-line reports are described in detail in D6.1.2.1 User interface specification. Software modules are however described here.

5.9.1 Online Reporting Tool – view and controller

The module AccessibilityObservatory implements the view and controller parts of the on-line reporting tool as a Plone product.

Location: robacc/UserInterface/GUI_Light/AccessibilityObservatory/

Module	***Description***
config.py	Configuration parameters for the Plone product.
configure.zcml	Configuration parameters for the Accessibility, such as choice between real data or fake data.
install-howto.txt	An installation description.
ReportSection.py	Functions used in the report section of the web page.
browser/	Folder containing python scripts for retrieving data for filling of the results tables of the page templates.
browser/BaseReport.py	Base class for the report view classes.
browser/ContentStatisticsReport.py	Content Statistics report view classes.
browser/DistributionReport.py	Distribution report view classes.
browser/HistoryReport.py	History report view classes.
browser/IndicatorsReport.py	Indicators report view classes.
browser/RegionalReport.py	Regional report view classes.
browser/WebsitesReport.py	Websites report view classes.
browser/mapping.py	Titles, short and long descriptions for all report types.
init.d/	Folder containing script to start the zope instance containing the Observatory interface at boot time.
interfaces/observatory_controller	Interface definition for the controller.
navmenu/	Folder containing code for the geographical/regional breadcrumbs.
profiles/	Folder containing profiles needed for automatic installation of the

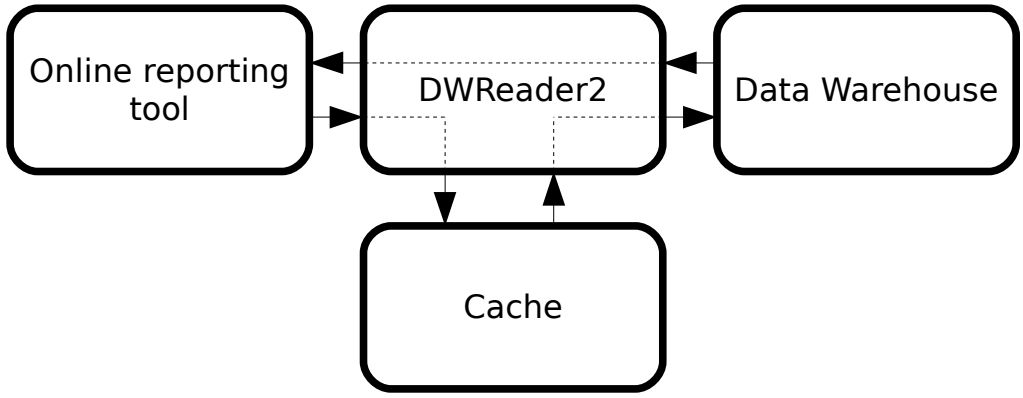
Module	***Description***
	AccessibilityObservatory Plone product.
skins/AccessibilityObservatory	Folder containing customised versions of some of the page template and css files from the plone skin.
templates/	Folder containing page templates for each of the report types.
tests/	Folder containing Unit tests for the AccessibilityObservatory.
utilities/	Folder containing utilities for retrieving real data from the Data Warehouse or fake data.
utilities/ObservatoryController.py	Controller that contains functions for retrieving and adapting data from the DW using the DWReader2 module.
utilities/FakeObservatoryController.py	Version of the controller that retrieves fake data instead of real data from the DW.
utilities/fakedata.py	Fake data for use in testing when DW is not available.
utilities/mapping.py	Mapping between NUTS/NACE codes and names.

5.9.2 DWReader2 - Data Warehouse GUI Model

The DWReader2 works as the model of the on-line reporting tool. In short it provides the Data Warehouse data to the controller-part of the on-line reporting tool.
Because of this the main functionality is to run predefined queries based on input functionality from the GUI controller, which again is based on input from the user requests. Furthermore, this module provides a cache of the data warehouse results (`DWResults`). Note that this is in addition to the already materialised views in the Data Warehouse. This cache has two purposes:

1. Reduce the computational load of the Data Warehouse, which is expected to occur if the online reporting tools is heavily used.
2. Avoid performance limitations in the online reporting tool for potentially slow Data Warehouse requests.

The work flow of the DWReader2 is outlined in Drawing 2. Any request from the reporting tool is sent to `DWReader2`. This module firsts performs a lookup in the cache. If the requested result already exists, the result is returned to the reporting tool without any Data Warehouse request. In contrast, if the requested result does not exist in the cache, a select statement is generated and a request sent to the Data Warehouse. The Data Warehouse result is written to the cache and returned to the Online reporting tool.



Drawing 2: Work flow of DWReader2

The cache uses the python `pickle` module and is synchronised to disk if the cache is updated. Writing the cache to disk allows the cached results to be available even if there is a restart of the online reporting tool. Note that all disk writing is done in a separate thread, and should because of this not influence the overall performance of the online reporting tool.

5.9.3 DWReader2 modules

Location: robacc/DWReader2

Module	*Description*
DWReader2.py	Main DWReader module including cache functionality. Creates queries to the Data Warehouse based on requests from the online reporting tool
DWError.py	Generates exceptions from the DWReader2 and Data Warehouse. Such exceptions include error connection to the Data Warehouse etc.
setup.py	Installation procedure
test.py	Unit tests
__init__.py	Package description

5.10 Other modules

5.10.1 INSTALL - Installation module

Installation script for Fedora Core 7.

Location: robacc/INSTALL

Module	*Description*
FC7Install.sh	Script for FC7 dependencies
INSTALL	Installation description

5.10.2 SystemConfiguration - System configuration module

System configuration variables are available in the System Configuration Module. Such variables include location of the WAM servers, which Data Warehouse database is used, etc.

All such variable need to be defined prior to running a test run and is intended to remain unchanged unless the Observatory setup changes. For parameters that are required to be part of the System Configuration see chapter 2.2.

All parameters should be defined in `initial.rdf` as RDF/XML, typically located in `/etc/eiao/initial.rdf`. The RDF is parsed using Redland and each variable is available as normal variable of the system configuration instance.

For reasons of clarity we present an example.

Update of parameters in initial.rdf

```
<rdf:Description rdf:about="http://www.eiao.net/rdf/systemconfiguration">
...
    <eiao:loglocation>/var/log/eiao/</eiao:loglocation>
...
</rdf:RDF>
```

Using the system configuration

```
>>> import sc
>>> sc = sc.SystemConfiguration()
>>> print sc.loglocation
/var/log/eiao/
```

Location: robacc/SystemConfiguration

Module	*Description*
sc.py	Main module
scerror.py	Exception classes for the System Configuration
setup.py	Installation procedures
test.py	Unit tests
__init__.py	Package definition
initial.rdf	Template for system configuration in /etc/eiao/initial.rdf
systemconfiguration_schema.rdf	RDF/OWL schema for system configuration
README-initial.rdf.txt	Brief explanation of system configuration options

5.10.3 Monitoring - Scripts for monitoring ongoing test runs

A working Observatory should run automatically without interaction from an administrator. However, as is the case with most large software application, monitoring the ongoing system is required to make sure that the Observatory is running correctly – especially during test phase. Note that in this chapter we only describe how to verify that the Observatory is running correctly. In order to locate a problem in a running Observatory, please refer to chapter 9.

An automatic script designed to do most of this work called `checkstatus.py`[39]:

39 Note that these processes may not run on the same machine. If the Observatory is running on more than one server, checkstatus.py needs to be run on more than each server.

The output is typically like the following:

```
The number of sites Crawler and loaded is: 1789
Sites left to crawl: 1093
Sites in sampling queue: 257

Number of crawlers running: 50
Number of samplers running: 5
Number of ETLs running: 1

Time since written to the Crawler log: 1.2 seconds.
Time since written to the Sampler log: 3.1 seconds.
Time since written to the ETL log: 38.2 seconds.

Checking WAMs
    http://someserver.com:8888/ OK
    http://anotherserver.com:8888/ OK
```

Location: robacc/Monitoring

Module	***Description***
checkstatus.py	Return statistics from EIAO crawler and sampler.

5.10.4 DBCleaner - Cleaning of logs and databases

Keeping old logs and RDF databases requires an enormous amount of disk space. This is only possible for smaller test runs during testing. It is not viable to keep such files during an actual run of the Observatory. Furthermore, MySQL has a limitation on the number of databases that is possible to create. Since the Observatory uses one RDF database per site, such limitations will be reached for larger crawls unless databases are moved / removed

For one site, there will in total be 6 individual log files in addition to the RDF database; standard output and standard error for the Crawler, Sampler and ETL.

To deal with this problem, a service has been created to handle cleaning (removing) of old logs and databases, hereby referred to as DBCleaner.

Logs and databases are cleaned in two separate ways:

(1) **Normal execution:**
Whenever the ETL is finished processing, the data for the site is inserted in the Data Warehouse, and the corresponding RDF-database is no longer needed. Because of this, the ETL sends a cleandb(database) to the DBCleaner. The DBCleaner then drops the RDF database and deletes all logs.

(2) **Dealing with special situations**
Some site results are never sent to the ETL, e.g.:
- sites that do not match the minimum requirement of 30 pages,

- sites that have been discarded due to problems such as segmentation fault in the Sampler[40].
In this situation (1) will never occur, and thus the DBCleaner will not be called, resulting in no cleaning of databases and logs.
To deal with this situation, the DBCleaner cleans up all logs and databases older than seven days on a daily basis. This means that any sampled results that has not reached the ETL within seven days are deleted[41].

Usage:

The DBCleaner module contains the `dbcleaner` server and the `dbcleanerclient` module, which can be used to remove old RDF databases. The client is normally being run as a cron job, and it removes RDF databases and logs older than 7 days, to avoid filling up the disks with RDF databases and logs. It also avoids problems with too many databases in MySQL.

Usage: `python dbcleanerclient database [OPTIONS]`
Options:
-p N - port to use. 8892 as default.
-s Server - Server to use. Localhost as default.
database - Database to remove (note remove all logs of this also)

Location: robacc/DBCleaner

Module	*Description*
dbcleaner	Main module
dbcleanerclient	Client to interact with the DB cleaner
removeoldlogs	Script to remove logs that at 7 days old
setup.py	Installation procedures

40 Note that these sites are re-sampled with a different database. I.e. the RDF database, logs etc. will be different.
41 The long duration of seven days is meant to deal with any latency in the system. E.g. if the ETL is busy, site results may be located in the queue for some time. However, a working Observatory is not expected to have a latency of seven days. Because of this, seven days should deal with any unexpected latency in the system.

5.10.5 CronWAM - Cron job for WAM service monitoring.

The module CronWAM contains a cron job `wamservicecheck` tailored for pure WAM machines. It checks every 10th minute that the WAM service is running and that it works properly, by sending a simple web page for evaluation that should return EARL. If EARL is not received, then the WAM is restarted.

Location: robacc/CronWAM

Module	*Description*
wamservicecheck	Checks that the WAM service on the local machine is running and that it returns EARL from a test HTML page.
setup.py	Installation script for cron job.

5.11 Utility functions

The Utility module contains many small utility functions. Some are written by the EIAO project and some come from other sources and are patched to work with EIAO.

5.11.1 GDB utilities

Utilities for debugging hang cases using GDB.

Location: robacc/Utilities/GDB

Module	*Description*
gdbworkaround.py	Program for inspecting the stack of all crawlers using GDB. The workaround was needed due to a problem in GDB which caused segmentation violation when inspecting the Python stack of each crawler thread on Fedora.
simpleanalysis.py	Analysis of the output of gdbworkaround.py to group crawlers being in different states.
examplehanging.py	Program that hangs to test gdbworkaround.py.

5.11.2 MemCache – 3rd party memcached client

Memcached is a client/server based distributed cache solution that is used by the crawler to cache information about crawled URLs.

Location: robacc/Utilities/MemCache

Module	*Description*
memcache.py	python-memcached client patched with timeoutsocket to avoid crawler hangs.
setup.py	Installation script
__init__.py	Package definition
README	Licence information about original file.

5.11.3 QA – Quality Assurance modules

Location: robacc/Utilities/QA

Module	*Description*
licensechecker.py	Reports files that does not have the GPL licence. (Note that 3rd party files must have their original licence.)
licence_header.txt	Standard GPL licence header for new files.
analysisrunner.py	Checks coding standard, unit test existence and profiling.
codingstandardchecker.py	Coding standard checker used by analysisrunner.py
repositoryfiles.py	Return all python files in the repository.

5.11.4 profiling – Source code profiling

Location: robacc/Utilities/profiling

Module	*Description*
timeprofile.py	Module for time profiling to optimise the run time of EIAO modules
profilechecker.py	Collects statistics from all profiling files
setup.py	Installation script
__init__.py	Package definition

5.11.5 eiaotime – Time module patch to avoid internal OS error 514

Location: robacc/Utilities/eiaotime

Module	*Description*
eiaotime.py	This will avoid Unknown error 514 for all time time.sleep(...) is called.
test.py	Unit tests.
__init__.py	Package definition.
setup.py	Setup script.

5.11.6 timeoutsocket - 3rd party patch to add timeout support for sockets

Location: robacc/Utilities/timeoutsocket

Module	*Description*
timeoutsocket.py	This module enables a timeout mechanism on all TCP connections. It does this by inserting a shim into the socket module.
setup.py	Installation script.
__init__.py	Package definition.

5.11.7 redland/unicodepatch – Redland for Python unicode patch

Location: robacc/Utilities/redland/unicodepatch/

Module	*Description*
RDF.py	Unicode patch for Redland.

5.12 Other data

5.12.1 EIAO RDF schema

This is the RDF schema with OWL restrictions that defines the RDF format that is generated by the Observatory and loaded into the data warehouse by the ETL.

Location: robacc/RDFSchema

Module	*Description*
RDFSchema.rdf	RDF schema for EIAO RDF.

5.12.2 Input URLs

This module contains scoped input URLs (seed URLs) for the Observatory.

Location: robacc/InputURLs

Module	*Description*
Capgemini/*	The Capgemini URL set categorised by country can be found under `robacc/InputURLs/Capgemini`. This is the uncleaned raw data.
CleanURLs/*	The module `robacc/InputURLs/CleanURLs` was used for initial conversion of the Capgemini URL set to the .csv format that can be imported to the EIAO Observatory.

5.12.3 Documentation

Documentation of the EIAO Observatory (user and system documentation).

Module	*Description*
EIAODocumentation.odt	Documentation of the EIAO Observatory (this document).

6 Future improvements

6.1 PDF WAM

The prototype PDF WAM, that detects tagged PDF and PDF version should be integrated into future versions of the Observatory.
If the project limits itself to not following PDF links, then PDF documents can be sampled directly from the URL repository, if the Crawler is configured to store PDF links. This would limit the effect of downloading PDF documents, since only the sampled documents need to be downloaded.

6.2 Animated GIF AWAM

We have created a very early proof-of-concept GIF WAM. It needs to be extended to measure the colour contrast thresholds required in UWEM. GIF images would need to be downloaded autonomously by the WAM, in the same way as CSS stylesheets currently are.

Note that GIF handling does not affect the speed of the initial breadth-first URL scan, since the crawler does not need to download the GIF images.

6.3 Crawler web page caching

The Crawler currently does not store downloaded web pages, because of space limitations and since the number of pages evaluated is relatively small compared to the set of pages initially scanned. This means that the sampled set of web pages will effectively be downloaded twice. Once during the URL scan crawl and once during the evaluation of a sample of web pages from the URLs crawled.

The performance effects of this can be limited by operating through a web cache (e.g. Squid), if the web cache has sufficient capacity. However for large scale operation, a web cache may have a negative effect since the web cache does not work well together with persistent HTTP connections.

A future improvement may therefore be to use HarvestMan's mirroring functions to store the downloaded web pages and let the Sampler access these directly on a site basis, to avoid downloading the sampled web pages twice.

2. Bibliography

[EIAO] European Internet Accessibility Observatory - http://www.eiao.net
[UWEM] Unified Web Evaluation Methodology version 1.2 - http://www.wabcluster.org
[WCAG1.0] Web Content Accessibility Guidelines version 1.0 - http://www.w3.org/TR/WAI-WEBCONTENT/
[SVN] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato Version Control with Subversion2004 - http://svnbook.red-bean.com/en/1.0/
[D3.3.2] Annika Nietzio WAM specification2008 - http://www.eiao.net/publications/
[SOAP] W3C SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)2007 - http://www.w3.org/TR/soap12-part1/
[EARL] Daniel Dardailler, Sean B. Palmer Evaluation and Report Language (EARL) 1.0 Schema2007 - http://www.w3.org/TR/EARL10-Schema/
[ISO19757-3] ISO Information technology - Document Schema Definition Languages (DSDL) Part 3: Rule based validation - Schematron2006 - http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip
[Xalan-J] Apache Software Foundation Xalan-Java version 2.7.12006 - http://xml.apache.org/xalan-j/
[BATIK] Apache Software Foundation Batik SVG Toolkit2008 - http://xmlgraphics.apache.org/batik/
[TestReport] Morten Goodwin Olsen, Nils Ulltveit-Moe Test report of version 2.2 of the Observatory2008 - http://www.eiao.net/
[REGEXP] Regular expression - http://en.wikipedia.org/wiki/Regular_expression
[HarvestMan] Anand B. Pillay HarvestMan - http://www.harvestmanontheweb.com/http://www.harvestmanontheweb.com/
[MEMCACHED] Brad Fitzpatrick memcached2008 - http://www.danga.com/memcached/
[RBN] Paul Sowden rbnarcissus - a javascript parser ported to ruby.2005 - p://idontsmoke.co.uk/2005/rbnarcissus/
[Redland] Dave Becket Redland RDF libraries - http://librdf.org/

[XMLCSS] James Clark Associating Style Sheets with XML documents1999 - http://www.w3.org/TR/xml-stylesheet/
[RELAX] Petr Nalevka The Relaxed HTML validator.2007 - http://www.relaxed.cz
[SCHEM] Information technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-based validation -- Schematron - http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip
[UWEM0.5] Unified Web Evaluation Methodology 0.5 - http://www.wabcluster.org/uwem05/
[WINPDB] Nir Aides Winpdb - A Platform Independent Python Debugger2008 - http://www.winpdb.org/
[GDB] Stephan Deibel DebuggingWithGdb2007 - http://wiki.python.org/moin/DebuggingWithGdb
[OWL] W3C OWL Web Ontology LanguageReference2004 - http://www.w3.org/TR/owl-ref/
[WWVALID] Sean Bechhofer and Raphael Volz WonderWeb OWL Ontology ValidatorAccessed 2006 - http://phoebus.cs.man.ac.uk:9999/OWL/Validator

7 Appendix A – Scope Pattern List Schema

```
<?xml version="1.0" encoding = "US-ASCII"?>


<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.wabcluster.org"
  xmlns="http://www.wabcluster.org"
  elementFormDefault="qualified">
  <!-- targetNamespace and default namespace are just
  placeholders... -->


  <xs:element name="scopePatternList">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="rule" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="type" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="include"/>
                  <xs:enumeration value="exclude"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="pattern" use="required" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>


</xs:schema>
```

The scope pattern list was first published in UWEM 0.5 [UWEM0.5].

8 Appendix B – Instance of scope pattern list

```
<?xml version="1.0" encoding = "US-ASCII"?>

<!--
      In the following example, the scope rules include everything from the
      site http://www.example.com/site1/ except
      http://www.example.com/site1/admin/. However, the parts of the sites
      under /admin/private/ and /admin/public/ should be included.
      Furthermore, any URL with the parameters ?
      type=notsosecret&isitsecret=no should be included.
      Note that the list of rules goes from the most specific to the most
      general.

      Validated with http://tools.decisionsoft.com/schemaValidate/
-->

<scopePatternList
  xmlns="http://www.wabcluster.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <rule
      type="include"
      pattern="http://www\.example\.com/site1/admin/private/.*"
    />

    <rule
      type="include"
      pattern="http://www\.example\.com/site1/admin/public/.*"
    />

    <rule
      type="include"
      pattern=".*?type\=notsosecret\&amp;isitsecret\=no.*"
    />

    <rule
      type="exclude"
      pattern="http://www\.example\.com/site1/admin/.*"
    />

    <rule
      type="include"
      pattern="http://www\.example\.com/site1/.*"
    />

</scopePatternList>
```

9 Appendix C - Developer hints and tips

This section shows various tips and hints for developers who wants to extend or debug the Observatory.

9.1 Command line options

All eiao services run as daemons and can be started / restarted by command line. Note that all services need to be installed for this to work.

Crawlers
To start crawlers run:
`/etc/init.d/crawlers start`

WAM Server
To start the WAM Server run:
`/etc/init.d/relaxedwam start`

ETL Server
To start the ETL server run:
`/etc/init.d/etlserver start`

Sampling server
To start the Sampling server run:
`/etc/init.d/samplingserver start`

Site URL Server
To start the Site URL server run:
`/etc/init.d/siteurlserver start`

Furthermore, the site URL server can be managed by command line through the command `eiaoassess`.

To populate the URL repository from .csv-files the following is done:
`eiaoassess -i /directory/with/input/csv/files`

For descriptions of csv-files see section 3.2.

To start a test run, the following is needed:

`eiaoassess -r test run -t table`

e.g.

`eiaoassess -r 200802 -t site`

where test run is an integer value indicating the current month such as 200802 and table is the table in the URL repository containing a list of all sites. Note that this starts a crawl towards all populated sites. Further note, that to start a test run, all daemons must be running and the system configuration up to date (see section 2.2).

9.2 Logs

Several logs are available for monitoring progress of the Observatory. Note that the locations of these logs are specified in the system configuration (chapter 2.2).

Crawler.log

Contains main information about the Observatory, including which sites that have started/finished crawling, which sites have been started/finished sampling and which sites have been started/finished loading to the Data Warehouse.
Furthermore, information about the individual pages downloaded, sampled and read by the ETL is sent to this log. Note that a running Observatory will output to this log frequently.

Starting crawl should look like the following:

```
Wed, 20 Feb 2008 19:31:28 +0000: Crawling site:
http://www.example.com.si to DB rdf_200802_1203532277_31_4952392
```

A started Sampler should look like the following:

```
Starting sampling of :http://www.example.com
rdf_200802_1203532277_31_4952392
```

A finished crawl and sample look like the following:

```
Wed, 20 Feb 2008 19:02:13 +0000: Finished crawling and sampling
site:http://www.example.com to db rdf_200802_1203532277_31_4952392
     Number of available scenarios reached from the Crawler 606
```

A started ETL should look like the following:

```
Currently handling website:http://www.example.com in the ETL
```

Note that no error messages are sent to this log.

Crawlers.log

Information about number of processess running and stopped from the crawlers daemon.

Samplingserver.log

Information about which which sites have been sent to the Sampler, which sites have finished and which sites are scheduled for sampling (including number of sites scheduled for sampling).

Etlserver.log

Information about which which sites have been sent to the ETL, which sites have finished and which sites are scheduled for loading to the Data Warehouse (including number of sites scheduled for loading to the Data Warehouse).

Siteurlserver.log

Information about which which sites that are scheduled for monitoring, which sites that have finished and which sites that have been sent to the crawlers (from site urls server) and number of sites left to crawl.

relaxedwam.log

Log messages from the Relaxed WAM container (python part including BWAM tests).

relaxed.log

Log messages from the Relaxed validator (AWAM handling, Java part).

Individual logs for each Crawler, Sampler and ETL

More detailed information of each individual process is available, that is not presented in the logs above. Such logs are useful to e.g. investigate crawl towards one site. Note that these logs in a standard installation are deleted when a site has reached the Data Warehouse. This is done to avoid storing huge logs which in most cases only is useful during debugging. Each run of a site is uniquely identified by the corresponding rdf-database. [42]

The following assumes that the site has been crawled to the database:
`rdf_200802_1203532277_31_4952392.`

Standard output and standard error from an individual Crawler is available at :
`crawler_df_200802_1203532277_31_4952392_stdout.log` and
crawler_ `rdf_200802_1203532277_31_4952392_stderr.log.`

They contain massive information about internal crawler state handing, discarded URLs, etc. Detailed understanding from the crawlers is required to understand these logs. If a Crawler crashes, the traceback will in most cases be available in:
crawler_ `rdf_200802_1203532277_31_4952392 _stderr.log.`

Standard output and standard error from the Sampler is available at:
`sampler_rdf_200802_1203532277_31_4952392_stdout.log` and
`sampler_rdf_200802_1203532277_31_4952392_stderr.log.`

They contain heavy information about internal Sampler behavior. Note that the standard error is not used in this component unless some error has been detected (in contrast to the Crawler), including errors that will appear occasionally such as problems downloading pages or timeout to the WAM.

Standard output and standard ETL from the Sampler is available at:
`etl_ rdf_200802_1203532277_31_4952392_stdout.log` and
`etl_ rdf_200802_1203532277_31_4952392_stderr.log.`

They contain information about internal etl behavior, including URLs handled or number of facts inserted to the Data Warehouse. Note that the standard error is not used in this

42 Note that if a site is crawled twice, individual databases will be used.

component unless some error has been detected (in contrast to the Crawler), including errors/warning that will appear occasionally such as unknown media type detected.

9.3 Debugging the Observatory

9.3.1 Starting ETL server in no-write (dummy) mode

The etl server can be started in no-write (dummy) mode, which means that the ETL server replies to store requests, but does not store the data:

```
python etlserver nowrite
```

This is useful for testing the Crawler and Sampler without actually storing the results in the Data Warehouse.

9.4 Remote Cpython debugger using Winpdb

The Observatory supports remote debugging of Crawler processes and threads via Winpdb/rpdb2 [WINPDB]. Remote debugging of running crawlers is an efficient way to manage and debug a full-scale running Crawler, since all crawlers then are configured as rpdb2 servers. Winpdb is not efficient for debugging hang cases and deadlock situations in the crawlers. If the crawlers hang, GDB with added python support should be used instead.

You will need to install Winpdb separately from http://www.winpdb.org.
When the Observatory is running in debug mode, `runCrawler2` will not poll the site URL server for hanging crawlers identified by lack of keepalive messages sent to the URL server, to avoid that the watchdog is killing crawlers being debugged.

You can find the code for the debug version of the Crawler here:

http://svn.eiao.net/robacc/Crawler/debugcrawler

A test program, using a fake Crawler is here:

http://svn.eiao.net/robacc/Crawler/debugcrawlerTest.py

Debugging support is enabled by calling runCrawler2.py with debug option, e.g:

python runCrawler2.py 100 3 0 debug

9.5 Cpython debugging using GDB

The GNU debugger (gdb) with added Python macros to inspect the cPython stack is the most efficient way of debugging IPC problems like deadlocks and other hang cases in the Crawler and Sampler, since it is able to break in to and inspect a running Python process in almost all cases. Procedures for installing GDB to run under Python can be found in [GDB].

It is important to install a version of cPython with debug symbols to be able to use GDB. On Fedora, use:

```
yum --enablerepo=core-debuginfo --enablerepo=updates-debuginfo
install python-debuginfo
```

Unfortunately, the GNU debugger hung whenever a python stacktrace was extracted from a running python application. In order for the GNU debugger to be useful for the project, a workaround using a combination of pexpect, threads that time out and GDB was developed. This is available here:
http://svn.eiao.net/robacc/GDB/gdbworkaround.py

It is run as following:

```
python gdbworkaround.py PID
```

The above will provide stacktrace for all threads within the python application for pid: PID.

9.6 Design rules

9.6.1 Design rule: Always use unicode internally.

Character set encoding problems easily arise if not a common encoding scheme is used. Several parts of the Crawler did not correctly support unicode encodings. By forcing unicode to all internal datastructures to use unicode, the problems can be fixed. Previously, the Crawler used encodings of the downloaded pages rather than unicode.

9.6.2 Design rule: Block instead of using active waiting

Avoid active waiting in multithreaded code, to improve responsiveness and reduce waisted CPU cycles.

An example of a not recommended implementation is:

```
while not data:
    data = queue.getnextdata()
    time.sleep(1 second)
```

The recommended implementation is:

```
data = queue.getnextdata() # Blocked until data is available.
```

9.6.3 Design rule: Parametrise all constant values.

In order to make the code of the Crawler more readable, all constant values should be substituted with variables. This makes the implementation a lot more readable, thus reducing the time of investigating the code during bug tracking.

For example:

```
try:
    data = downloaded_page()
except 404Error:
    result = -1
else:
   result = 0
```

Instead:

```
DOWNLOAD_OK = 0
DOWNLOAD_NOT_OK = -1
try:
    data = downloaded_page()
except 404Error:
    result = DOWNLOAD_NOT_OK
else:
   result = DOWNLOAD_OK
```

10 Appendix D Content statistics WAM (MWAM)

Content statistics data like technologies, language or CSS media types are harvested by AWAM modules. The content statistics data is categorised by content statistics WAMs (MWAMs), that read the relevant meta data from the AWAMs, and produce instances of the eiao:MetaData RDF class extending the EARL report sent from the WAM. The eiao:MetaData class instances refer to earl:TestSubject, and are reported in a similar way as earl:Assertion.

10.1 Updated RDF ontology

An updated EIAO OWL ontology [OWL] for version 2.0 can be found at http://www.eiao.net/rdf/2.0. The ontology validates with the WonderWeb ontology validator [WWVALID] as OWL Full.

This ontology is used in regression testing to verify that the RDF graphs conform to the OWL restrictions defined in the ontology.

10.2 eiao:MetaData RDF class

The eiao:MetaData RDF class contains these properties:

Meta data	RDF class	RDF property name	RDF type	#Instan-ces
Reference to web page being tested.	eiao:MetaData	earl:subject	string	1
Describes the type of content statistics.	eiao:MetaData	eiao:metaDataType	string	1
Describes the value of the content statistics (depending on type).	eiao:MetaData	eiao:value	string	1
Describes the location (line, column) of the content statistics.	eiao:MetaData	eiao:singleLocation	eiao:singleLocation	1

Table 13: Properties of MetaData RDF class.

The rdf:ID property is be set to a unique ID by the WAM. The WAM container constructs a globally unique ID based on the WAMs own unique ID concatenated with “-M” and a running sequence number per WAM.

Example: Assuming that a WAM named MyWAM is defined, then this wam would generate unique RDF Ids for meta data of the form:

```
<eiao:MetaData rdf:ID=”MyWAM-M1065183”>
```

10.2.1 eiao:MetaDataType RDF class

The eiao:type class determines the type of content statistics that is being conveyed. It is referenced by the eiao:type property in eiao:MetaData. The following types have been defined:

Type	Label	Description
eiao:language	Language	If eiao:type is eiao:language, then eiao:value conveys content statistics about the natural language used in the web site.
eiao:technology	Technology	If eiao:type is eiao:technology, then eiao:value conveys content statistics about technologies used to render this web site. Technologies are described using the MIME type[43], e.g. text/html, application/pdf or image/gif.
eiao:internalLinks	Internal links	If eiao:type is eiao:internalLinks, then eiao:value conveys content statistics about technologies this web page links to within the assessed web site. Internal links are described using the MIME type, e.g. text/html, application/pdf or image/gif.
eiao:externalLinks	External links	If eiao:type is eiao:externalLinks, then eiao:value conveys content statistics about techologies this web page links to outside the assessed web site. External links are described using the MIME type, e.g. text/html, application/pdf or image/gif.
eiao:mediaType	CSS media type	If eiao:type is eiao:mediaType, then eiao:value conveys a list of output media types that this web page supports[44]. (E.g. all, aural, braille, embossed, handheld, print, projection, screen, tty, tv)
eiao:contentFilter	Content pre-filter applied	If eiao:type is eiao:contentFilter, then eiao:value contains the name of the content prefilter that was applied. Currently, the only value will be “tidy”, in case HTML Tidy was used to fix fatal parse errors in the HTML.

Table 14: Types of content statistics supported by the eiao:MetaData class.

Example of use:

```
<eiao:type rdf:about=”#technology”/>
```

43 MIME types are registered by IANA: http://www.iana.org/assignments/media-types/
44 Recognised CSS media types are defined by W3C: http://www.w3.org/TR/REC-CSS2/media.html

10.2.2 eiao:value property

The eiao:value property of the eiao:MetaData class describes the value of the content statistics, in the context given by eiao:type.

Example:

```
<eiao:MetaData rdf:ID=”MyWAM-M5423452”>
      .
      <eiao:type rdf:about=”#technology”/>
      <eiao:value>application/x-shockwave-flash</eiao:value>
      .
      .
</eiao:MetaData>
```

10.2.3 eiao:rangeLocation class

The eiao:rangeLocation property is the same class that we currently use to describe the location of an evaluation result in the EARL. This class defines the position as (line,column) like this:

```
<eiao:SingleLocation rdf:ID="L342178">
        <eiao:line>13</eiao:line>
        <eiao:column>28</eiao:column>
</eiao:SingleLocation>
```

10.2.4 RDF report showing the eiao:MetaData extension

```
<earl:TestSubject rdf:about="http://www.example.org/index.html">
  <dc:title xml:lang="en">Example web page</dc:title>
  <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#gDate">2006-03-16
11:18+0100</dc:date>
</earl:TestSubject>

<eiao:SingleLocation rdf:ID="MyWAM-L342178">
        <eiao:line>13</eiao:line>
        <eiao:column>28</eiao:column>
</eiao:SingleLocation>

<eiao:MetaData rdf:ID=”MyWAM-M3409870”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#language”/>
      <eiao:value>nb_NO</eiao:value>
      <eiao:singleLocation rdf:resource="#MyWAM-L342178”>
</eiao:MetaData>
```

Note that the WAM report the language tags as they are encoutered, so that two letter language codes like. `<eiao:value>en</eiao:value>` are also possible.

10.3 Technology usage statistics

MIME type	Technologies used to compose web page	Technologies linked internally in web site	Technologies linked externally (to other web sites)
text/html	1 per page	4 per page	1 per page
text/css	2 per page		
application/x-javascript	4 per page		
application/msword		0.2 per page	
application/pdf	0.2 per page	1 per page	
Etc...			

Table 15: Example of statistics on technology usage for a given web site.

Technology usage statistics will be reported as MIME types in three different types of eiao:MetaWam:

Category	Description
`<eiao:type rdf:about="#technology"/>`	Technologies used to compose web page
`<eiao:type rdf:about="#internalLinks"/>`	Technologies linked to within web site
`<eiao:type rdf:about="#externalLinks"/>`	Technologies linked to outside web site

Table 16: Technology statistics types.

The reason for splitting up the technology statistics into three categories, is that content/technologies a web page links to is treated differently from content that a web page embeds to render a web page. Content a web page links to belongs to other web resources, which at some time in the future may be evaluated by the Observatory[45]. Links within the web site may be more relevant relevant statistics for a given web site than links pointing to other web sites, so it makes sense to separate technologies identified by these two categories of links.

45 Currently we assess only (X)HTML and CSS documents. Other documents are not opened for performance reasons.

10.3.1 Technologies within a web page

Technologies within a web page will be identified by the AWAMs in the section “Technologies used within the web page” in table 17 in D3.3.2.

Each instance of the technologies identified by these AWAMs will be reported as an eiao:MetaWam like this example shows (assuming a PDF was found):

```
<eiao:MetaData rdf:ID=”MyWAM-M9830245”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#technology”/>
      <eiao:value>application/pdf</eiao:value>
      <eiao:singleLocation rdf:resource="#L3429734”>
</eiao:MetaData>
```

10.3.2 Technologies a web page links to

Technologies within a web page are identified by the AWAMS in the section “Technologies a web page links to” in table 17 in D3.3.2.

Link direction identification will be performed by comparing the base of the identified URL to the base URL of the test subject. If the link URL is http://www.google.com/search.html and the test subject URL is http://www.example.com/test.html, then the MWAM module would compare www.google.com to www.example.com. Since these domains differ, then the link would be marked as an external link. Otherwise it would be marked as an internal link.

Internal link; i.e. base URL of test subject == base URL identified by AWAM is reported as:

```
<eiao:MetaData rdf:ID=”MyWAM-M6452345”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#internalLink”/>
      <eiao:value>application/pdf</eiao:value>
      <eiao:singleLocation rdf:resource="#L3429734”>
</eiao:MetaData>
```

External link; i.e. base URL of test subject != base URL identified by AWAM is reported as:

```
<eiao:MetaData rdf:ID=”MyWAM-M5342452”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#externalLink”/>
      <eiao:value>application/pdf</eiao:value>
      <eiao:singleLocation rdf:resource="#L3429734”>
</eiao:MetaData>
```

10.4 Content filters (e.g. HTML Tidy) applied

Content filtering is identified by the AWAMs in the section “Parsing with HTML Tidy” in table 17 in D3.3.2.

If HTML Tidy has been applied, to make the content possible to parse for the Relaxed based AWAM, then eiao:MetaData with eiao:type contentFilter and eiao:value “Tidy” will be created:

```
<eiao:MetaData rdf:ID=”MyWAM-M5342452”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#contentFilter”/>
      <eiao:value>Tidy</eiao:value>
      <eiao:singleLocation rdf:resource="#L3429734”>
</eiao:MetaData>
```

10.5 Language

Language content statistics indicators are defined in the section “General properties of the web page” in table 17 of D3.3.2.

The language-handling module for identifying language checks, in order of precedence, the following AWAMs:

- EIAO.A.10.4.3.1.HTML.2.1 lang attributes
- EIAO.A.0.0.0.2.HTML.1.1 Language identifiers in META elements
- EIAO.A.10.4.3.1.HTTP.1.1 HTTP Content-Language

Each instance of language attributes or elements identified by the WAMs above will be reported as eiao:MetaData instances in the EIAO EARL report as shown in the example below, assuming language pl_PL:

```
<eiao:MetaData rdf:ID=”MyWAM-M4552341”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#language”/>
      <eiao:value>pl_PL</eiao:value>
      <eiao:singleLocation rdf:resource="#L3421578”>
</eiao:MetaData>
```

Note that the Data Warehouse also needs to accept two letter ISO-639 codes, because it is not possible to reliably extend the two letter codes with country codes.

10.6 CSS media types

CSS media type content statistics indicators are defined in the section “General properties of the web page” in table 17 of D3.3.2.

Media type	Occurrences per page
screen	1 per page
braille	0.6 per page

Table 17: Example of statistics on CSS media types for a given web site

Statistics on CSS media types are important to tell which output media a given web page supports. This may also reveal information on whether a particular web site has been adapted for accessibility needs (e.g. Aural or Braille output supported.)

Each instance of CSS media types will produce an eiao:MetaData instance like shown below, assuming media type print has been identified:

```
<eiao:MetaData rdf:ID=”MyWAM-M3124533”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#mediaType”/>
      <eiao:value>print</eiao:value>
      <eiao:singleLocation rdf:resource="#L2421578”>
</eiao:MetaData>
```

The media types are reported as they are found in the CSS files. This means that e.g. mediatype all, which usually is the default media type, is reported as:

```
<eiao:MetaData rdf:ID=”MyWAM-M2125434”>
      <earl:subject rdf:resource=”http://www.example.org/index.html”>
      <eiao:type rdf:about=”#mediaType”/>
      <eiao:value>all</eiao:value>
      <eiao:singleLocation rdf:resource="#L2421578”>
</eiao:MetaData>
```

10.7 How to infer MIME types

10.7.1 How to infer MIME types from file extension

These AWAMs will return URIs, and not the MIME type of the technology directly:

- EIAO.A.0.0.0.2.HTML.10.1 Image element
- EIAO.A.0.0.0.2.HTML.11.1 Hyperlink elements
- EIAO.A.0.0.0.2.HTML.4.1 – URI of code <object> without codetype
- EIAO.A.0.0.0.2.HTML.9.1 – URI of data <object> without type

For these AWAMs we infer the MIME type of image objects based on file extension for performance reasons.

The mime type of the referenced image is inferred from the file name extension of the image[46] using the Python mimetypes library, as shown below:

```
>>> import mimetypes
>>> mimetypes.guess_type("http://www.example.org/bilde.gif",strict=1)
('image/gif', None)
```

The first element in the tuple specifies the MIME type, and the second element specifies an optional encoding.

10.7.2 Inferring MIME type for applet element

EIAO.A.0.0.0.2.HTML.5.1 identifies the deprecated applet element, which has a mime type set to application/x-java-applet.

10.8 Measurement data handling

Measurement data is data about one or more measurements performed in the Observatory, and some measurement data is precalculated and conveyed to the Data Warehouse to offload the computational load of the ETL.

Measurement data may be generated on 4 levels:

- Site level (eiao:siteSurvey)
- Web page (or RDF Page scenario) level (eiao:scenario)
- Test subject level (earl:TestSubject) e.g. a single web page
- Test result level (earl:TestResult) most detailed level.

46 Note that there exist some CMS frameworks that don't use file extension to indicate the image type, but instead stores object type information as properties in an object database. One well known example is the Zope application server. However, it's still common practice to use file extensions even for these systems, since many authoring tools require known file extensions to work properly.

Measurement data is conveyed as RDF properties to existing RDF classes in EARL or the EIAO RDF scheme.

In general, all data from the HTTP header will be passed on to the AWAM, so that both barrier indicator reporting WAMs (BWAMs) and content statistics WAMs (MWAMs) can use these AWAM metadata in their calculations.

10.8.1 eiao:siteSurvey measurement data

Meta data	RDF class	RDF property name	RDF type	#Instances
Site average barrier indicator over all samples *Fs*	eiao:siteSurvey	eiao:barrierIndicator	float	1
Standard deviation *Ss* for barrier indicator over all samples	eiao:siteSurvey	eiao:variance	float	1
Error margin	eiao:siteSurvey	eiao:errorMargin	float	1
URL count (URL repository snapshot)	eiao:siteSurvey	eiao:urlCount	int > 0	1

Table 18: Measurement metadata and warnings on site survey level.

Data on site survey level will be calculated or extracted by the Sampler. The data consists of measurement data on site level (*Fs,Ss and error margin*). The URL count may be used as an indicator on whether the Crawler is able to crawl the web site properly or not.

The example below shows how eiao:siteSurvey is extended with the new properties:

```
<eiao:siteSurvey rdf:about="http://www.eiao.net/rdf/1.0#x.fr.st_survey_91">
       <eiao:barrierIndicator>0.56</eiao:barrierIndicator>
       <eiao:variance>0.21</eiao:variance>
       <eiao:errorMargin>0.05</eiao:errorMargin>
       <eiao:urlCount>600</eiao:urlCount>
       <eiao:pageSurvey
       rdf:resource="http://www.eiao.net/rdf/1.0#x.fr.st_pageSurvey_161"/>
       <eiao:scenario
       rdf:resource="http://www.eiao.net/rdf/1.0#x.fr.st_survey_91_scenario_274"/>
       .
       .
       <eiao:basedownloaddir>
       /var/local/cache/eiao/harvestman/storedfiles/EIAO_SC/91/le-tampon.fr.st
       </eiao:basedownloaddir>
       <eiao:webSite>http://le-tampon.fr.st/</eiao:webSite>
</eiao:siteSurvey
```

10.8.2 eiao:scenario meta data (web page level)

Meta data	RDF class	RDF property name	RDF type	#Instances
Page aggregated barrier indicator *Fp*	eiao:scenario	eiao:barrierIndicator	float	1

Table 19: Measurement data on web page (RDF page scenario) level.

The aggregated barrier indicator *Fp* is a property of the eiao:scenario class and describes the aggregated barrier indicator for one web page. This parameter is added by the Crawler to the RDF page scenario structure when the RDF scenario structure is written. The formula for how to calculate *Fp* is described in D3.3.2.

The example below shows an example of use of eiao:barrierIndicator in eiao:scenario:

```
<eiao:scenario rdf:about="#x.y.net_survey_92_scenario_282">
      <eiao:barrierIndicator>0.67</eiao:barrierIndicator>
      <eiao:typeofscenario>pagescenario</eiao:typeofscenario>
      <eiao:rangeLocation
        rdf:resource="#x.y.net_survey_92_scenario_282_range-20776"/>
</eiao:scenario>
```

10.8.3 Content statistics data for earl:TestSubject

Meta data	RDF class	RDF property name	RDF type	#Instances
Calculated content length	earl:TestSubject	eiao:contentLength	int	1

Table 20: Measurement meta data on test subject level.

We currently store information about ContentLength in the HTTP header. However this content length may not always be available. The Crawler therefore in addition calculates the string length in octets of the received object, and stores this in the eiao:contentLength property.

Example of use:

```
<earl:TestSubject rdf:about="http://x.y.be/prix_marcel_thiry.htm">
      <dc:title>Automatically sampled by European Internet Accessibility
Observatory</dc:title>
      <dc:date>2006-03-01 14:58 +0100</dc:date>
      <eiao:contentLength>2344</eiao:contentLength>
</earl:TestSubject>
```

10.8.4 earl:result measurement data

Meta data	RDF class	RDF property name	RDF type	Cardinality (RDF instances)
Rpb parameter in D3.2.1	earl:result	eiao:barrierIndicator[47]	float	1

Table 21: Measurement meta data on test result level.

To convey the *Rpb* barrier indicator parameter, the EARL generating library (earl.py) adds the property eiao:barrierIndicator to the class earl:result. This allows for the possibility for BWAM modules that produce a continuous value between 0 and 1, instead of only pass and fail in the future.

```
<earl:result rdf:about="r1141220976r2305">
      <earl:validity rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-
      strawman#fail"/>
      <eiao:barrierIndicator>0.8</eiao:barrierIndicator>
</earl:result>
```

11 Appendix E Terms and acronyms

Term	Use in EIAO
AAU	University of Aalborg
Accessibility barrier indicator *Fp*	*Fp* indicates an accessibility barrier for a given web page.
Accessibility test	A test referring to an evaluation of one or more web pages resulting in an EARL report.
Aggregation	Grouping of data to get an overview of the result set.
AWAM	Analytical WAM (Analyses HTML and CSS, and returns indicators and their location.)
Balanced Scorecard	Approach to strategic management developed by Robert Kaplan and David Norton. The approach can be used to monitor progress towards a defined set of goals.
BenToWeb	Project in the WAB cluster, focusing on three special Accessibility

47 According to the latest EARL working draft, the earl:confidence property might have been used instead of eiao:barrierIndicator, since the workgroup now has dropped the requirement to have 3 levels (high, medium, low). However, the semantics of this parameter is still open, so we chose to define a specific parameter that suited our needs. Our EARL with the EIAO extensions is backwards compatible with EARL, so that other tools will be able to read EARL reports from the WAMs. However they will not understand our meta-data extensions in the EIAO RDF schema.

Term	Use in EIAO
	test modules
BWAM	Barrier composing WAM. (Composes a statement about accessibility based on one or more AWAM indicators.)
CP	Checkpoint
Crawl web site	Recursively download web pages, until a termination criterion terminates the crawling process.
CSS	Cascading Style Sheets
CSS2	Cascading Style Sheets, level 2
CWAM	Composing WAM (Aggregates BWAM results for e.g. a web site or a region to a single value.)
DOM	Document Object Model
DW	Data Warehouse
EARL	Evaluation And Reporting Language
EIAO	European Internet Accessibility Observatory
ERT	Evaluation and Repair Tools
ETL	Extract, Transform and Load. ETL refers to the process of getting data out of one data store (extract), moving and transforming it (transfer), and inserting it into a different data store (load).
FTB	Forschungsinstitut Technologie Behindertenhilfe (Research institute Technology, Aids)
GIF	Graphics Interchange Format
HTML	Hyper Text Markup Language
IPR	Intellectual Property Rights
Materialised views	Precomputed results of database queries that are stored in the database and used by the database system for faster query processing.
MMU	Manchester Metropolitan University
NACE	Acronym from the French 'Nomenclature statistique des Activites economiques dans la Communaute Europeenne'- Statistical classification of economic activities in the European Community
NUTS	Nomenclature of Territorial Units for Statistics
Page sampling	Page sampling is the process of selecting a representative web pages from a site..
Python	Programming language used for core parts of the Observatory
RDF	Resource Description Framework
Repository	Electronic store of structured information. e.g. used for storing web pages, metadata, URLs, and test results
Scoping	Scoping is the process of focusing an accessibility test to a certain area of the web.
Seed resource	A starting point for crawling a web site.
SOAP	Simple Object Application Protocol
SQL	Structured Query Language – the standard relational database query language

Term	Use in EIAO
UiA	University of Agder
URI	Uniform Resource Identifiers (URL + URN) [http://www.example.com/foodir/foofile]
URL	Universal Resource Locator [http://www.example.com/foodir/]
URN	Universal Resource Name [foofile]
User testing	Evaluation by use of a representative set of users
UWEM	Universal Web Evaluation Methodology
W3C	World Wide Web Consortium
WAB	Web Accessibility Benchmark cluster
WAI	Web Accessibility Initiative
WAM	Web Accessibility Metrics. Property of a web page or a web site that can be assessed and that is expected to have an impact on accessibility. Also used more specifically for a test definition based on an automatically testable requirement or sub requirement derived from a WCAG 1.0 checkpoint.
WAM server	A server providing Web Accessibility Metrics evaluation services for the EIAO . Returns EARL reports.
WCAG	Web Content Accessibility Guidelines
Web resource	A network data object or service that can be identified by a URI. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, resolutions) or vary in other ways.
Web site	A collection of interlinked Web pages, including a host page, residing at the same network location
WP	Work Package
XHTML	The Extensible Hyper Text Markup Language
XML	Extensible Markup Language