Katholieke
Universiteit
Leuven

**Department of
Computer Science**

PROJECT REPORT

DESIGN OF SOFTWARE SYSTEMS

Daniel Andrés Pérez Pérez
Jasper Marien
Robin Vanden Ecker
Romain Carlier

Contents

1 Design decisions

two pages

1.1 Iteration 1 & 2: Image Filtering

1.2 Iteration 3: Content Filtering

We separated the distinct responsibilities among several (object) classes. *FilterHttpContent* is the link to the restof ZAP. It extends *FilterAdaptor* in the parosproxy filter package. Upon receiving an HTTP response, it only verifies the *HttpMessage* has content. Then it instantiates a *FilterApplyer* (concrete subclass) and calls one of its filtering methods with 2 parameters. Currently only one is available, but other methods can be added to offer various filtering algorithms. The 2 parameters are the *HttpMessage* and the url of the file containing the filter terms and additional info. The *FilterApplyer* then instantiates 2 helpers. One, a *PageContent*, represents a given (upon creation) *HttpMessage*'s content. Subclasses provide the content with the desired type, *String* for the assignment. The other, a *FormatFileToFilterInfo*, parses the file at given url and returns the useful information. Currently only one parsing method is available, that supports the format of the assignment and returns a *Pair*. This pair consists of the weight threshold and a list of *InappropriateElement* instances. *InappropriateElement* models inappropriate content, of generic type, its weight and explanatory tags. To support different formats, other parsing methods can be added. Next, the *FilterApplyer* combines both result and filters the content using the found inappropriate elements. An alternative with lower coupling would be by sending a closure from one class to the next one and never returning results.

1.3 Iteration 4: Content Reporting

It was created a new package to be consistent with the naming that zaproxy has. The package *org.zaproxy.zap.extension.imgreport* encapsulates the classes used for the extension. The class *ExtensionImageReport* extends *ExtensionAdaptor* (creates, initializes and hooks a new extension), *XmlReporterExtension* (gets our XML format which will be added to the zaproxy report extension) and *HttpSenderListener* (converts our new extension in an observer object which is able to catch all the HttpMessages).
ExtensionImageReport validates whether *HttpMessage* content is an image content, stores *HttpImage* objects and delegate the creation of specific statistics format to the *ImageStatistics* classes.
ImageStatisticsFactory instantiates new concrete classes of *ImageStatistics*.
HttpImage processes *HttpMessage* and returns the corresponding object.
ImageDimensionStatistics is an template class implementation of *ImageStatistics* used by *ImageHeightStatistics*, *ImageSizeStatistics* and *ImageWidthStatistics*. *ImageTypeStatistics* is an implementation of *ImageStatistics* that creates a unique XML format.
This *ExtensionImageReport* was implemented as core functionality since the given XML format to the ReportExtension relies on XSL style sheets to add the new information in HTML and MarkDown reports thus those corresponding XSL files were properly updated.

1.4 Iteration 5: Refactoring

1.4.1 org.parosproxy.paros.core.proxy.ProxyThread

Response was created to handle the errors messages while*notification* package was created to delegate all the notification method used in *ProxyThread*. Due to the similarity in the algorithm, it was implemented using an abstract template class and the internal behavior was implemented in the concrete classes.

2 Strengths of the design

one page

2.1 Iteration 1 & 2: Image Filtering

2.2 Iteration 3: Content Filtering

Splitting responsibilities increased cohesion of these classes. *FilterHttpContent* only handles messages now and does a basic check before sending the work to the *FilterApplyer*. This one coordinates the preparative tasks and does the actual filtering with the collected results. Then it returns the result to *FilterHttpContent*, but it could also set it itself. The only class with relatively high coupling is *FilterApplyer*. Due to the high cohesion, the classes are easy to understand and modify. Support for various extension is also provided with some generic typing and abstract super classes.

2.3 Iteration 4: Content Reporting

The extension is encapsulated in its package and relies in the interfaces provides by zaproxy. Due to the Strategy pattern applied for the image statistics, developers can create new concrete classes either using the template class or implementing a new one. Developers can select specific image statistics types via the *ImageStatisticsFactory*.

2.4 Iteration 5: Refactoring

2.4.1 org.parosproxy.paros.core.proxy.ProxyThread

Due to the template class *ProxyListenerNotifier*, developers can create new notification method using the concrete class without affecting the behavior of the others. The *notification* package can also be reused in other parts of the code since it does not depends on *ProxyThread*.

3 Weaknesses of the design

one page

3.1 Iteration 1 & 2: Image Filtering

3.2 Iteration 3: Content Filtering

3.3 Iteration 4: Content Reporting

The extension can add the new images statistics to the XML report in a straightforward way but it is not the case for HTML and MarkDown reports which are highly coupled to the XSL files. Whenever new XML image statistics format is created in the Image-Extension, the XSL files must be modified; the main issue is that those classes/files are not even directly related, making difficult to convert this *ImageExtension* into an add-on plugin.

3.4 Iteration 5: Refactoring

3.4.1 org.parosproxy.paros.core.proxy.ProxyThread

ProxyThread still has to create the concrete notification classes and stores them in a data structure. Hence the responsibilities were turned from calling internal methods to managing *ProxyListenerNotifier* classes.

4 Future improvements

one page

4.1 Iteration 1 & 2: Image Filtering

4.2 Iteration 3: Content Filtering

4.3 Iteration 4: Content Reporting

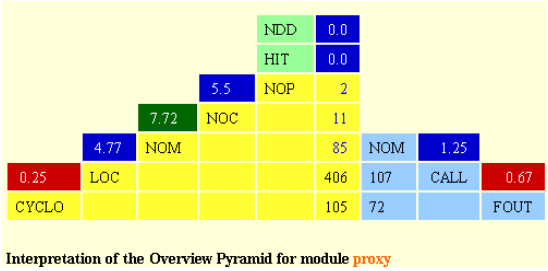
The current implementation does not allow the final user to select the specific image statistic type in the report. Next improvement considers a GUI implementation using the *hook* abstract method provide by *ExtensionAdaptor*. *ImageStatisticsFactory* can be adapted to add those responsibilities: keep tracking the final user image statistic type selections and instantiating/removing the corresponding ImageStatistics classes in runtime.

4.4 Iteration 5: Refactoring

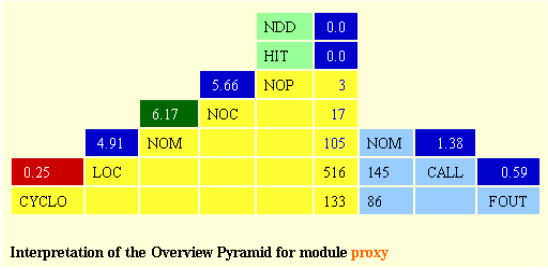
5 Overview pyramid and test coverage

one page

5.1 Pyramid



(a) Pyramid before refactoring



(b) Pyramid after refactoring

Figure 1: Pyramids of the org.parosproxy.paros.core.proxy package before and after refactoring.

5.2 Test Coverage

6 Design Diagrams

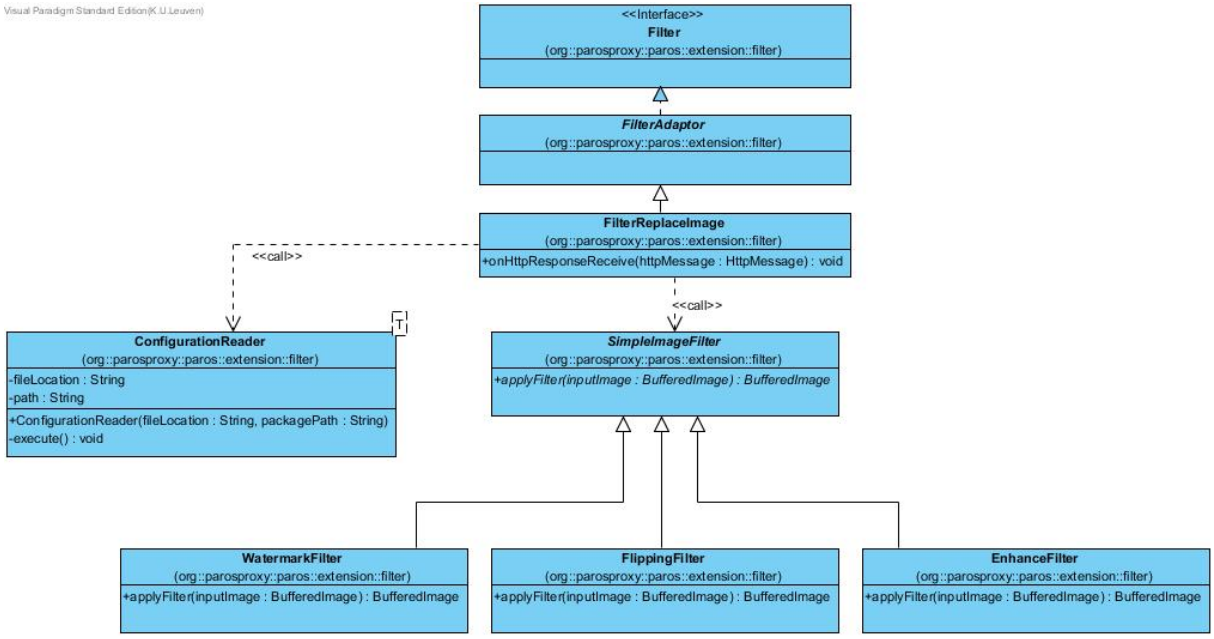


Figure 2: Class diagram iteration 1 & 2

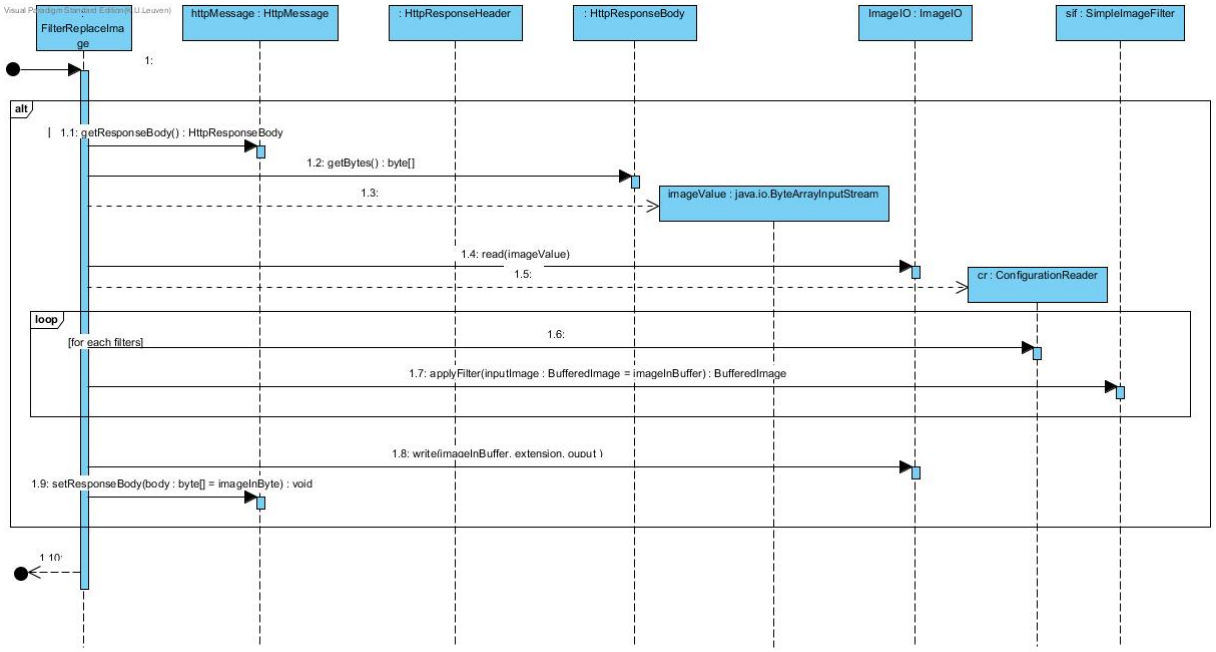


Figure 3: Sequence diagram of iteration 1 & 2

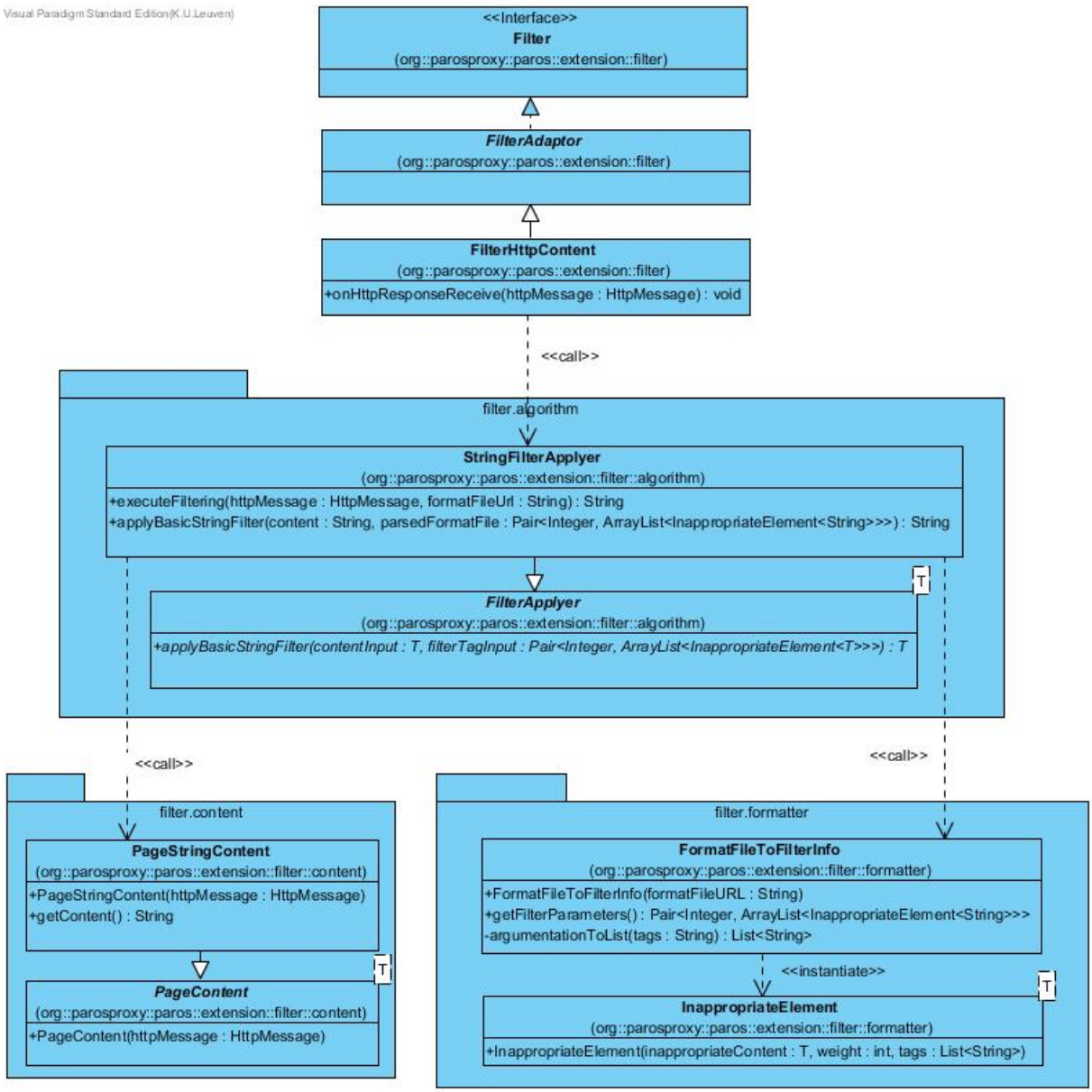


Figure 4: Class diagram of iteration 3

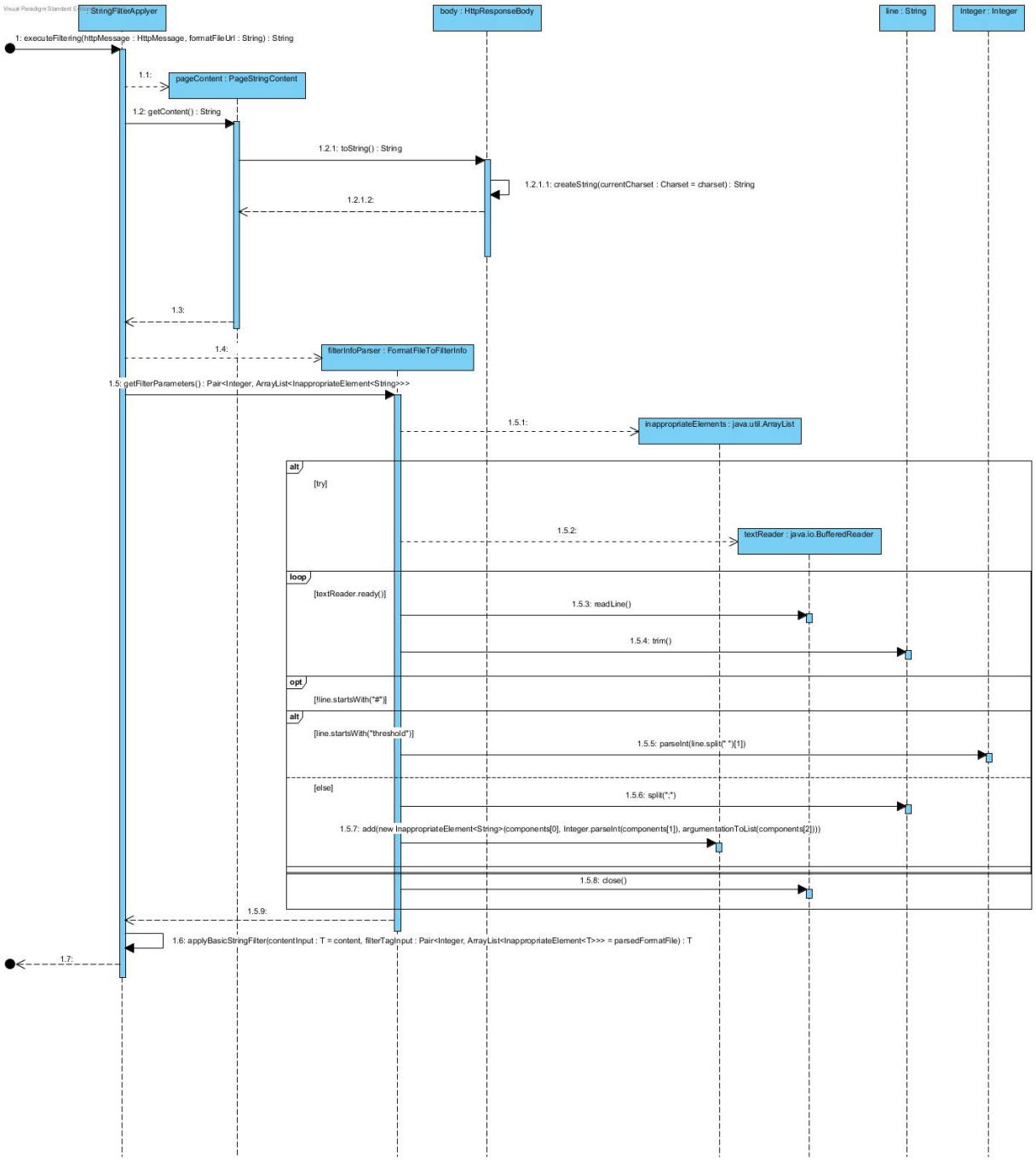


Figure 5: Sequence diagram of iteration 3

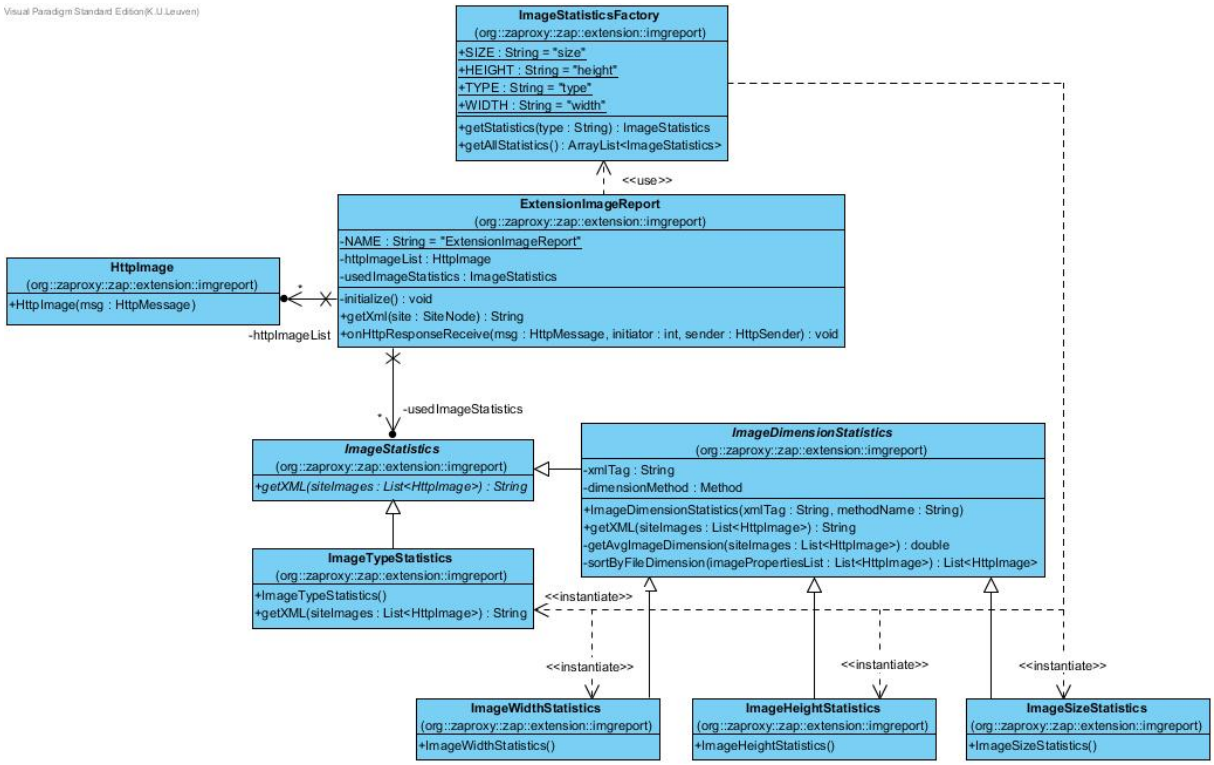


Figure 6: Class diagram of iteration 4

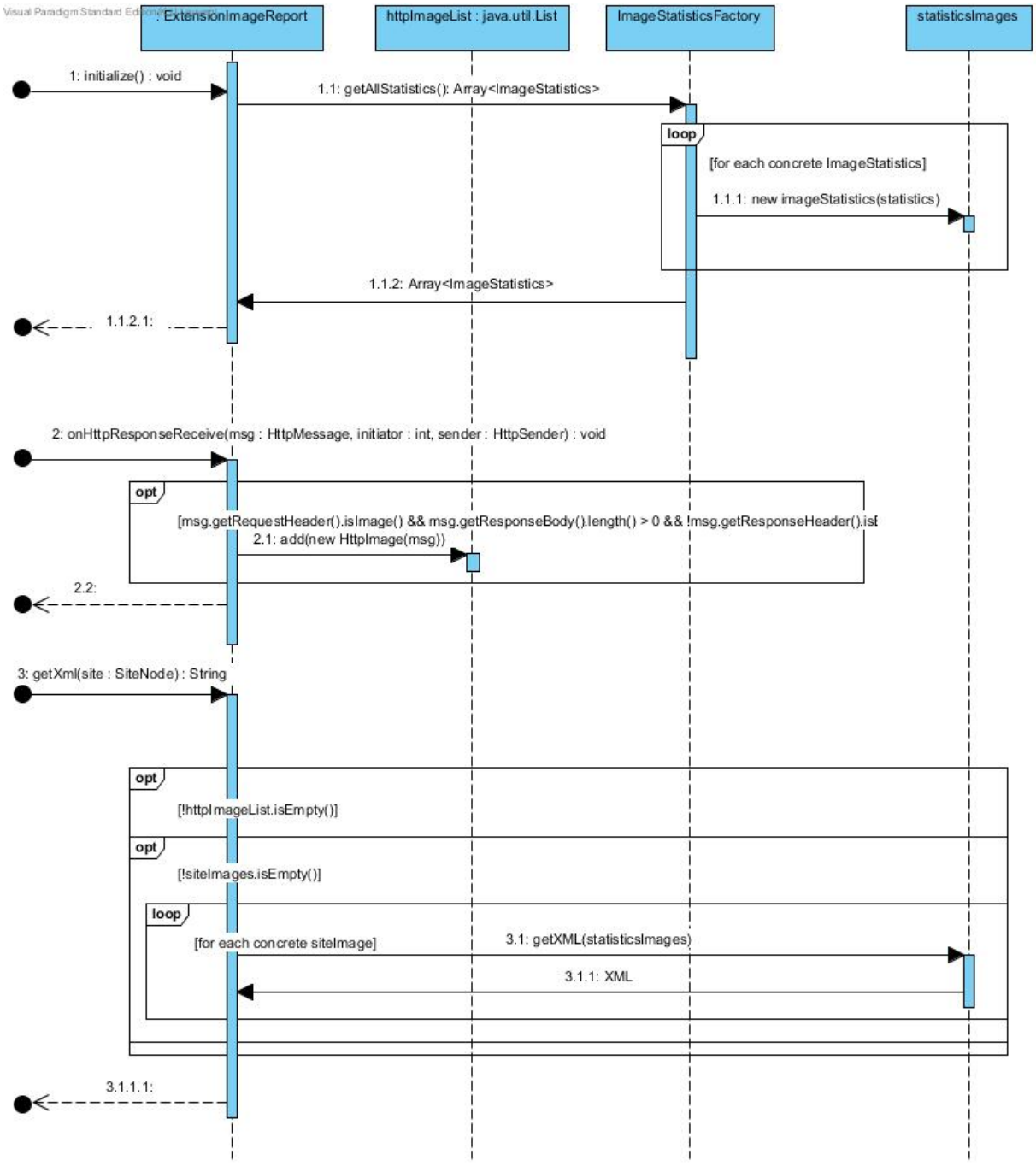


Figure 7: Sequence diagram of iteration 4