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

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 main *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 instantiates and stores new concrete classes of *ImageStatistics*, validates whether *HttpMessage* content is an image content, stores *HttpImage* objects and delegate the creation of specific statistics format to the *ImageStatistics* classes.
HttpImage processes *HttpMessage* and returns the corresponding object.
ImageDimensionStatistics is an implementation of *ImageStatistics* that contains a XML template format used by *ImageHeightStatistics*, *ImageSizeStatistics* and *ImageWidthStatistics*. *ImageTypeStatistics* is an implementation of *ImageStatistics* that creates a unique XML format.
This ImageExtension 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

2.3 Iteration 4: Content Reporting

The extension is encapsulated in its package and only 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 also remove specific type of statistics easily.

2.4 Iteration 5: Refactoring

We also made a minor improvement to methods going over all extensions. They now use a for-each (over the private list) rather than an indexed for-loop and the public 'getExtension(i)'. The order will be preserved so the behavior is the same.

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 ImageExtension, 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* to 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*. This implies the creation of a new *ImageStatisticsMaganer* class which responsibilities will be keeping track the final user image statistic type selections and instantiating/removing the corresponding ImageStatistics classes in runtime. *ExtensionImageReport* will call this manager to get the XML images statistics formats. A new call indirection is created while the image statistics tasks in *ExtensionImageReport* are delegated to *ImageStatisticsMaganer*, hence the responsibilities and cohesion are maintain in expert classes.

4.4 Iteration 5: Refactoring

5 Design Diagrams

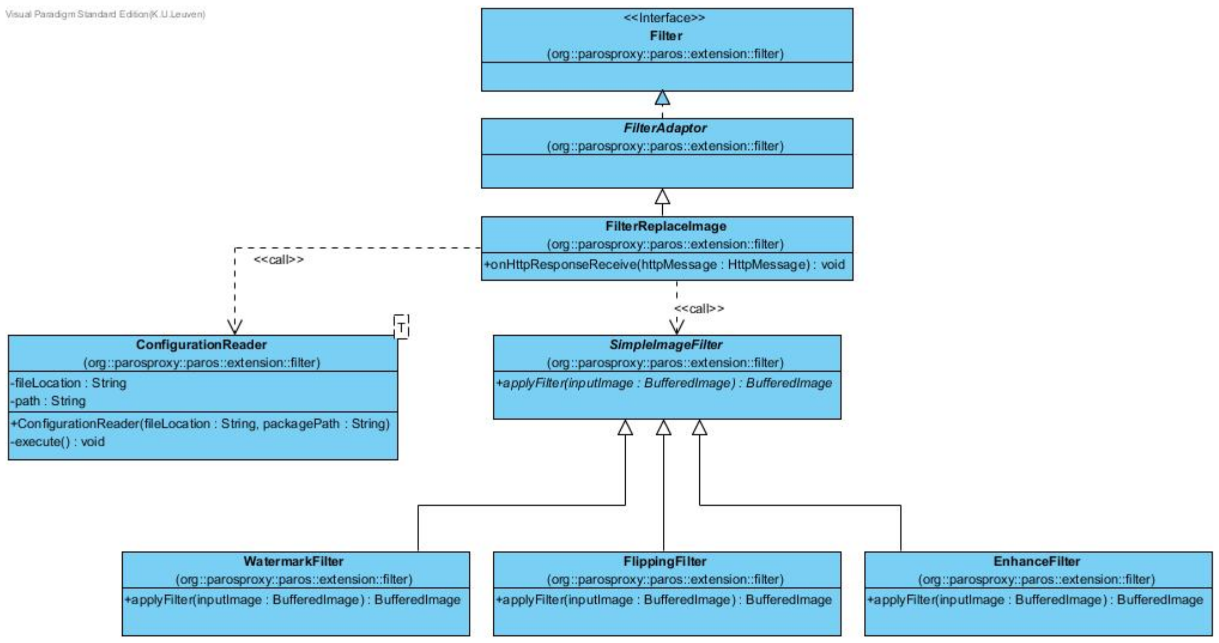

Visual Paradigm Standard Edition(K.U.Leuven)
<<Interface>>
Filter
(org::parosproxy::paros::extension::filter)
FilterAdaptor
(org::parosproxy::paros::extension::filter)
FilterReplaceImage
(org::parosproxy::paros::extension::filter)
+onHttpResponseReceive(httpMessage : HttpMessage) : void
<<call>>
<<call>>
ConfigurationReader
(org::parosproxy::paros::extension::filter)
-fileLocation : String
-path : String
+ConfigurationReader(fileLocation : String, packagePath : String)
-execute() : void
T
SimpleImageFilter
(org::parosproxy::paros::extension::filter)
+applyFilter(inputImage : BufferedImage) : BufferedImage
WatermarkFilter
(org::parosproxy::paros::extension::filter)
+applyFilter(inputImage : BufferedImage) : BufferedImage
FlippingFilter
(org::parosproxy::paros::extension::filter)
+applyFilter(inputImage : BufferedImage) : BufferedImage
EnhanceFilter
(org::parosproxy::paros::extension::filter)
+applyFilter(inputImage : BufferedImage) : BufferedImage

Figure 1: Class diagram iteration 1 & 2

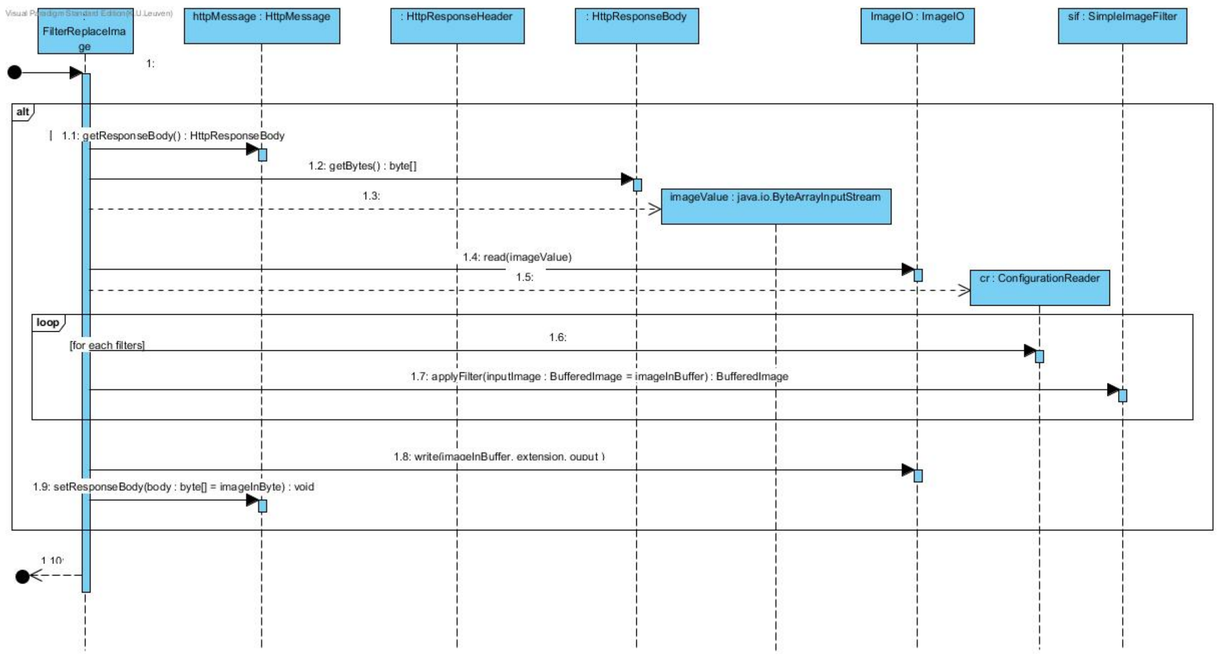
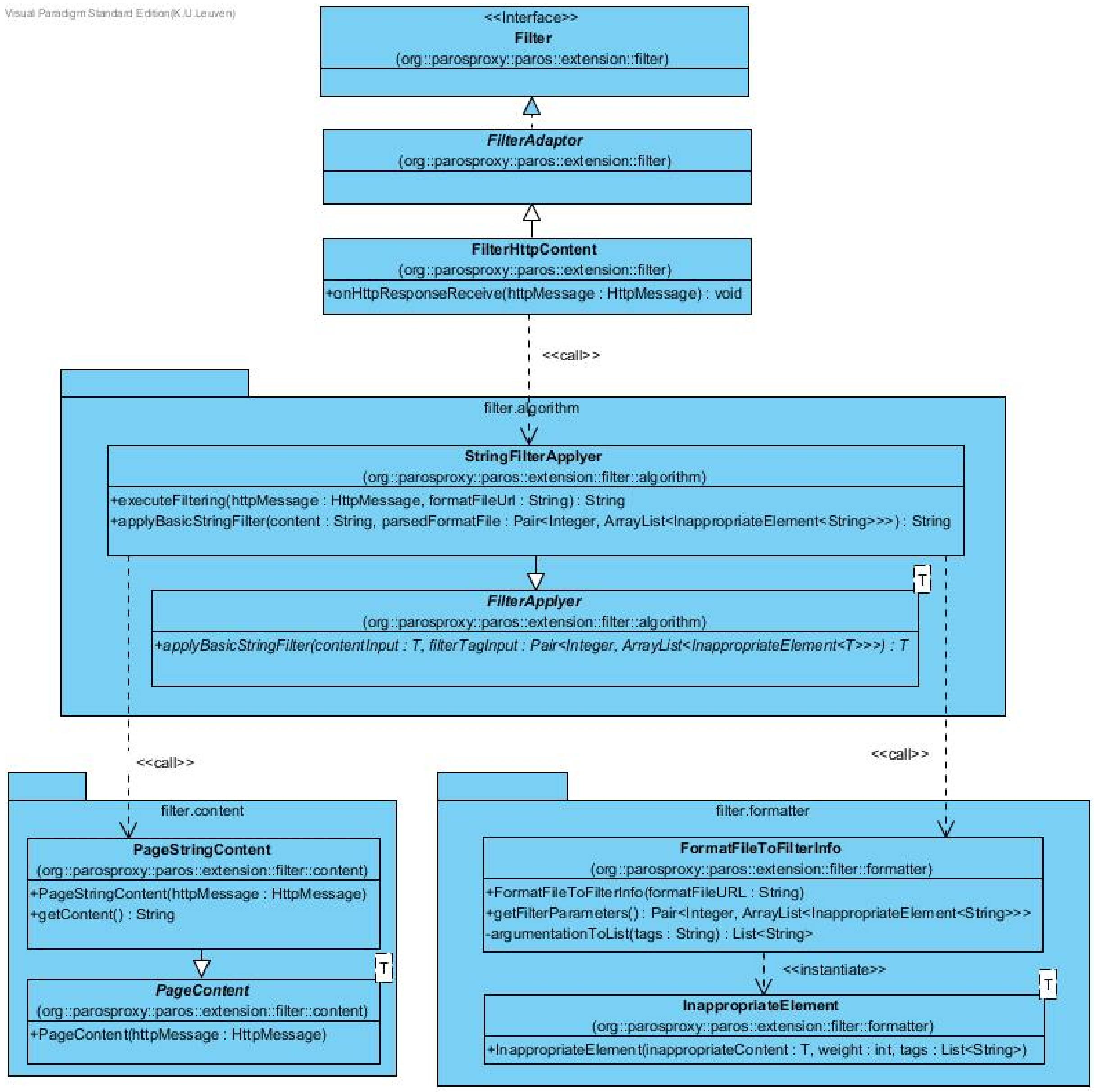

Visual Paradigm Standard Edition(K.U.Leuven)
FilterReplaceImage
httpMessage : HttpMessage
: HttpResponseHeader
: HttpResponseBody
ImageIO : ImageIO
sif : SimpleImageFilter
1:
alt
1.1: getResponseBody() : HttpResponseBody
1.2: getBytes() : byte[]
1.3:
imageValue : java.io.ByteArrayInputStream
1.4: read(imageValue)
1.5:
cr : ConfigurationReader
loop
[for each filters]
1.6:
1.7: applyFilter(inputImage : BufferedImage = imageInBuffer) : BufferedImage
1.8: write(imageInBuffer, extension, ouput)
1.9: setResponseBody(body : byte[] = imageInByte) : void
1.10:

Figure 2: Sequence diagram of iteration 1 & 2

6 Overview pyramid and test coverage

one page


Visual Paradigm Standard Edition(K.U.Leuven)
<<Interface>>
Filter
(org::parosproxy::paros::extension::filter)
FilterAdaptor
(org::parosproxy::paros::extension::filter)
FilterHttpContent
(org::parosproxy::paros::extension::filter)
+onHttpResponseReceive(httpMessage : HttpMessage) : void
<<call>>
filter.algorithm
StringFilterApplyer
(org::parosproxy::paros::extension::filter::algorithm)
+executeFiltering(httpMessage : HttpMessage, formatFileUrl : String) : String
+applyBasicStringFilter(content : String, parsedFormatFile : Pair<Integer, ArrayList<InappropriateElement<String>>>) : String
T
FilterApplyer
(org::parosproxy::paros::extension::filter::algorithm)
+applyBasicStringFilter(contentInput : T, filterTagInput : Pair<Integer, ArrayList<InappropriateElement<T>>>) : T
<<call>>
<<call>>
filter.content
PageStringContent
(org::parosproxy::paros::extension::filter::content)
+PageStringContent(httpMessage : HttpMessage)
+getContent() : String
T
PageContent
(org::parosproxy::paros::extension::filter::content)
+PageContent(httpMessage : HttpMessage)
filter.formatter
FormatFileToFilterInfo
(org::parosproxy::paros::extension::filter::formatter)
+FormatFileToFilterInfo(formatFileURL : String)
+getFilterParameters() : Pair<Integer, ArrayList<InappropriateElement<String>>>
-argumentationToList(tags : String) : List<String>
<<instantiate>>
T
InappropriateElement
(org::parosproxy::paros::extension::filter::formatter)
+InappropriateElement(inappropriateContent : T, weight : int, tags : List<String>)
Figure 3: Class diagram of iteration 3

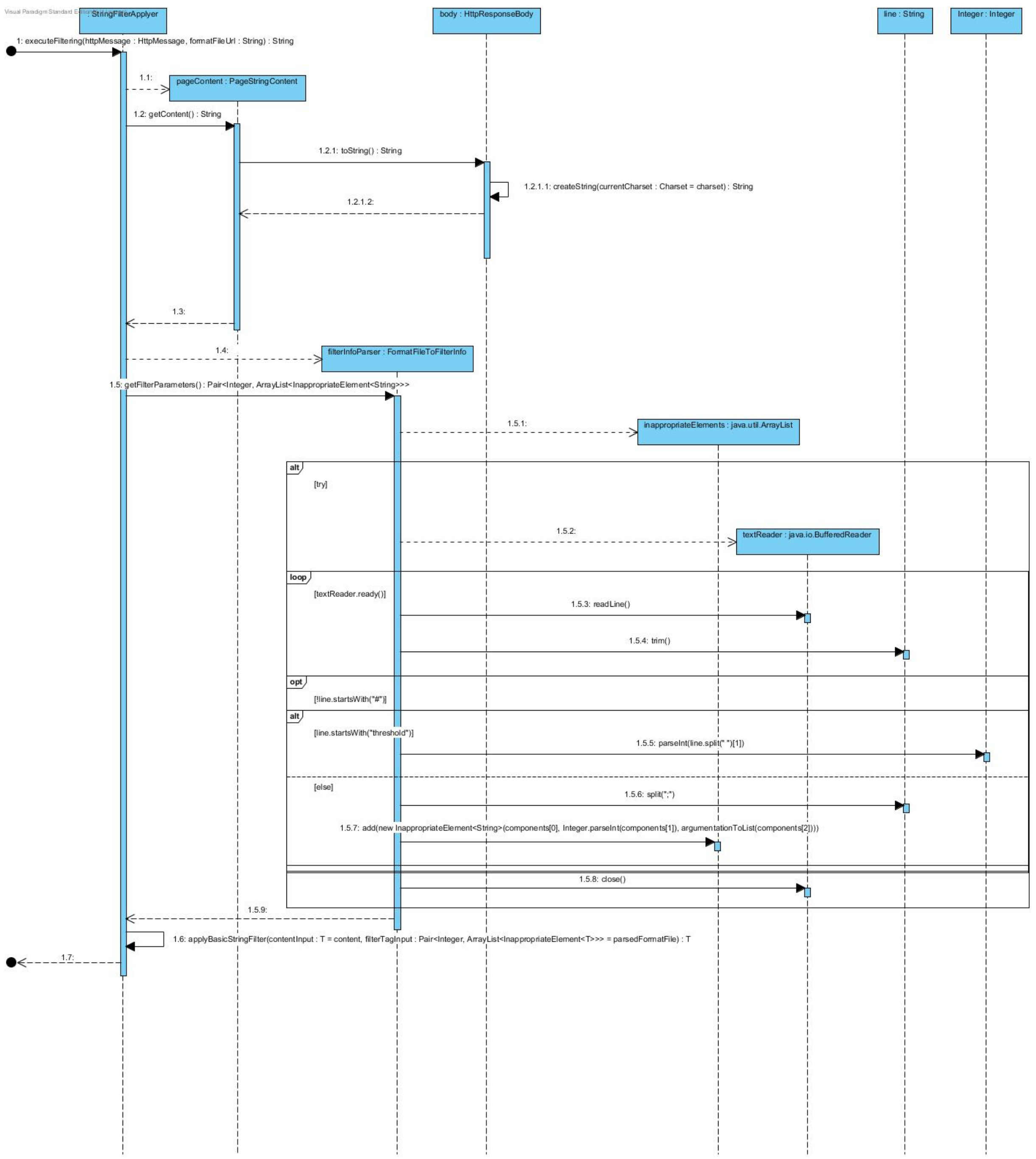

: StringFilterApplyer
body : HttpResponseBody
line : String
Integer : Integer
1: executeFiltering(httpMessage : HttpMessage, formatFileUrl : String) : String
1.1:
pageContent : PageStringContent
1.2: getContent() : String
1.2.1: toString() : String
1.2.1.1: createString(currentCharset : Charset = charset) : String
1.2.1.2:
1.3:
1.4:
filterInfoParser : FormatFileToFilterInfo
1.5: getFilterParameters() : Pair<Integer, ArrayList<InappropriateElement<String>>>
1.5.1:
inappropriateElements : java.util.ArrayList
alt
[try]
1.5.2:
textReader : java.io.BufferedReader
loop
[textReader.ready()]
1.5.3: readLine()
1.5.4: trim()
opt
[!line.startsWith("#")]
alt
[line.startsWith("threshold")]
1.5.5: parseInt(line.split(" ")[1])
[else]
1.5.6: split(";")
1.5.7: add(new InappropriateElement<String>(components[0], Integer.parseInt(components[1]), argumentationToList(components[2])))
1.5.8: close()
1.5.9:
1.6: applyBasicStringFilter(contentInput : T = content, filterTagInput : Pair<Integer, ArrayList<InappropriateElement<T>>> = parsedFormatFile) : T
1.7:

Figure 4: Sequence diagram of iteration 3

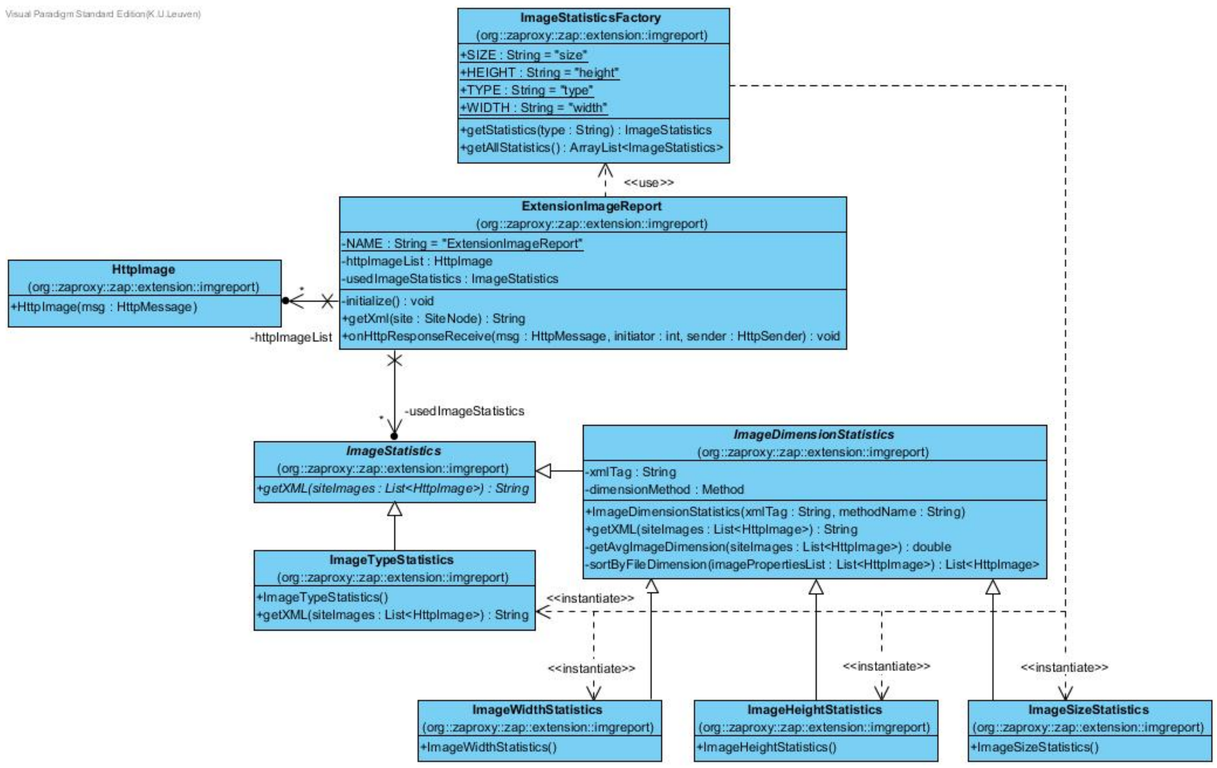

Visual Paradigm Standard Edition(K.U.Leuven)
ImageStatisticsFactory
(org::zaproxy::zap::extension::imgreport)
+SIZE : String = "size"
+HEIGHT : String = "height"
+TYPE : String = "type"
+WIDTH : String = "width"
+getStatistics(type : String) : ImageStatistics
+getAllStatistics() : ArrayList<ImageStatistics>
<<use>>
ExtensionImageReport
(org::zaproxy::zap::extension::imgreport)
-NAME : String = "ExtensionImageReport"
-httpImageList : HttpImage
-usedImageStatistics : ImageStatistics
-initialize() : void
+getXml(site : SiteNode) : String
+onHttpResponseReceive(msg : HttpMessage, initiator : int, sender : HttpSender) : void
HttpImage
(org::zaproxy::zap::extension::imgreport)
+HttpImage(msg : HttpMessage)
*
-httpImageList
-usedImageStatistics
*
ImageStatistics
(org::zaproxy::zap::extension::imgreport)
+getXML(siteImages : List<HttpImage>) : String
ImageDimensionStatistics
(org::zaproxy::zap::extension::imgreport)
-xmlTag : String
-dimensionMethod : Method
+ImageDimensionStatistics(xmlTag : String, methodName : String)
+getXML(siteImages : List<HttpImage>) : String
-getAvgImageDimension(siteImages : List<HttpImage>) : double
-sortByFileDimension(imagePropertiesList : List<HttpImage>) : List<HttpImage>
ImageTypeStatistics
(org::zaproxy::zap::extension::imgreport)
+ImageTypeStatistics()
+getXML(siteImages : List<HttpImage>) : String
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
ImageWidthStatistics
(org::zaproxy::zap::extension::imgreport)
+ImageWidthStatistics()
ImageHeightStatistics
(org::zaproxy::zap::extension::imgreport)
+ImageHeightStatistics()
ImageSizeStatistics
(org::zaproxy::zap::extension::imgreport)
+ImageSizeStatistics()

Figure 5: Class diagram of iteration 4

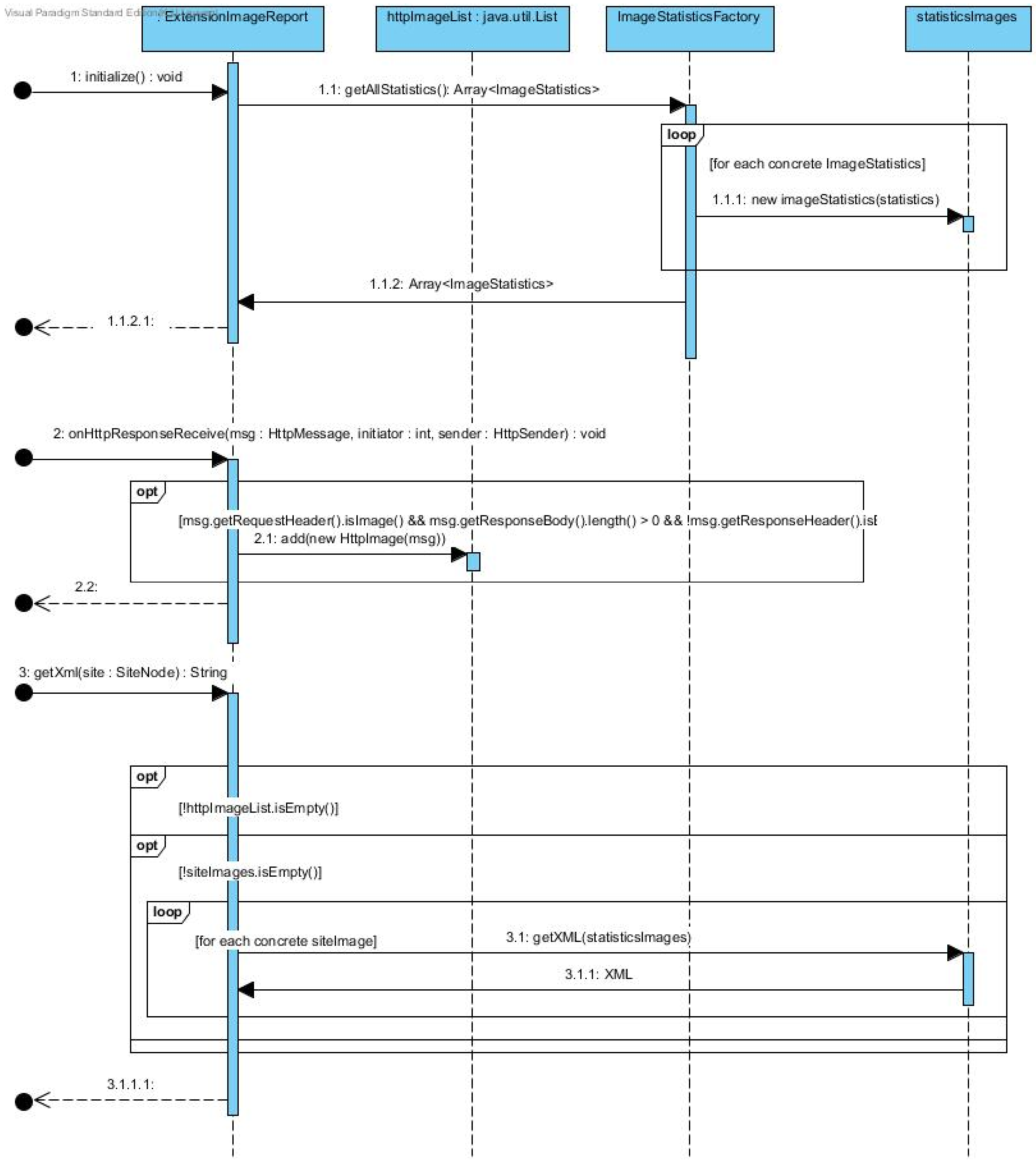

Visual Paradigm Standard Edition
: ExtensionImageReport
httpImageList : java.util.List
ImageStatisticsFactory
statisticsImages
1: initialize() : void
1.1: getAllStatistics(): Array<ImageStatistics>
loop
[for each concrete ImageStatistics]
1.1.1: new imageStatistics(statistics)
1.1.2: Array<ImageStatistics>
1.1.2.1:
2: onHttpResponseReceive(msg : HttpMessage, initiator : int, sender : HttpSender) : void
opt
[msg.getRequestHeader().isImage() && msg.getResponseBody().length() > 0 && !msg.getResponseHeader().is
2.1: add(new HttpImage(msg))
2.2:
3: getXml(site : SiteNode) : String
opt
[!httpImageList.isEmpty()]
opt
[!siteImages.isEmpty()]
loop
[for each concrete siteImage]
3.1: getXML(statisticsImages)
3.1.1: XML
3.1.1.1:

Figure 6: Sequence diagram of iteration 4