Ambient Earth

Design Document — 1st draft

Christian Luijten
christian06@ru.is

October 6, 2006

CADIA group
School of Science and Engineering
Reykjavík University, Reykjavík

Under supervision of:
Kristinn R. Thórisson, Ph.D. (Reykjavík University)
dr.ir. Huub van de Wetering (Eindhoven University of Technology)

Abstract

This document describes the design of the Ambient Earth system. Ambient Earth is a software system for visualization of Internet activity.

Contents

1 Introduction

This document describes the design of the Ambient Earth project. The ambitional goal of this project is to give an ambient view on the activity on the whole world-wide web. In practice, it shows the activity on for instance a forum or larger weblog system.

The name of the system is AMBER.

The design of the project will follow the Constructionist Design Methodology for Interactive Intelligences[Thó04]. First, a few usage scenarios are given in Chapter 2. These scenarios result in the requirements which are listed in Chapter 3. Using the requirements, an architecture is written up in Chapter 4.

Please note that in this document some basic knowledge of the terminology of Psyclone framework is assumed. For more information about Psyclone, refer to the documentation at `http://www.cmlabs.com/psyclone/manual/`.

2 Usage scenarios

2.1 A story is posted to a weblog

When a story is posted on a weblog, it will show up in its RSS feed (this happens of course outside of our responsibility). If AMBER is monitoring this particular weblog, it will retrieve the story and analyze it. The story is then displayed on a screen using the results of the analysis.

To get a more concrete idea, suppose AMBER is monitoring various A.I. related weblogs and we would like to find out what they are mainly writing about. We configure the analysis component in such a way that it can decide whether a certain subject is dealt with in a story, thereby creating a profile for every story. Stories with similar subjects will then show up close to eachother in the display, stories with orthogonal subjects will be very far apart.

The result is an image with various "clouds" of in some way related stories.

2.2 A discussion is held on a web forum

Discussions on web forums can get lengthy and the main subject can change multiple times during their lives. To get an idea what subjects the whole discussion has been about, AMBER can show a cloud map of (part of) the discussion. It could even show an animation to show how the discussion developed over time.

Using the animated view of the discussion development, a new participant in the discussion can decide upon whether bringing up an old discussion point is a good idea or not. It is also a way to locate a certain subject within a long list of replies.

3 Requirements

There are various kinds of requirements to be identified. A distinction can be made between functional and extra-functional (or non-functional) requirements.

3.1 Extra-functional requirements

1. The system must make use of the Psyclone framework for communication.
2. The system will be implemented in the Java programming language.
3. The display module with the Java Applet must be able to run on any machine with a properly installed Java Virtual Machine (i.e. not only on the machine running the rest of the system).
4. It must be possible to add modules with similar functionality to operate in parallel with modules already there. For example when the Java Applet is running, it should also be possible to have the full screen module running at the same time.

3.2 Functional requirements

These requirements describe which *inputs*, *outputs*, *storage* and *computations* exist in the system and how they are *timed and synchronized*. Finally, since this is a very important part of the project, there are two separate sections on *story analysis* and *visualization* requirements.

3.2.1 Inputs

1. The system must be configurable to specify which sources will be monitored.
2. The system will use the configuration to get information from the internet from the specified sources.
3. Configuration of the system goes via Psyclone using module parameters.
4. The display may have a set of controls to navigate through the history of a feed.
5. Parts of the system must accept triggers from Psyclone whiteboards.
6. Sources must be Rich Site Summary (RSS) feeds. The system should however be prepared to support other source types as well (i.e. it should be easily expandable).
7. The Applet display is non-interactive (no input).

3.2.2 Outputs

1. There is an output module which is to be used within a website, i.e. a Java Applet.
2. There is an output module which runs standalone and in full screen and displays more information than the Applet can.

3.2.3 Storage

1. The system on itself does not store anything.

3.2.4 Computations

1. The system must decide of a delivered story what its subject(s) is/are.
2. The system may put weights on the subjects instead of a boolean value.

3.2.5 Timing and synchronization

Synchronization between modules is handled by Psyclone, so no requirements need to be added to the system itself.

3.2.6 Story analysis

1. When stories come in, they are analysed by analysis modules.
2. Every module adds some meta-information to the story depending on the module analysis.

3.2.7 Visualization

The following requirements are common for both the applet and the standalone viewer.

1. A story is represented as a dot.
2. In the center of the display is Earth (with picture?).
3. Dots are launched into orbit around Earth.
4. The orbits follow Kepler's laws of planetary motion.
5. The launch velocity is dependent on how "big" the story is (like from an important author or if it has many references) at the time of writing, but is always smaller than the escape velocity.
6. Stories get velocity boosts by getting replies/comments or references ("trackbacks") in order to keep them around longer.

7. Stories which don't get reactions will just orbit Earth for a while and eventually fall back down (i.e. launch velocity < escape velocity).
8. Only the meta-information added to the stories by the Sieve modules is used to determine launch variables.
9. There are some small, heavy bodies in geostationary orbit around Earth representing values of an enumeration of meta-information (for instance story subjects). They attract the stories depending on how much they match the story's subject. It is possible for a story to get into orbit with such a heavy body if it is really strongly connected to the subject.

There will be two different views, a static and a dynamic one. Which one is used depends on the application. To get an idea of the activity at a certain moment in time, the static view is used. For a "real-time" view of internet activity, the dynamic view can be used.

The term "static" doesn't mean the image is standing still, it will behave exactly the same as the dynamic view. However, some physical laws don't apply or are differently calibrated in order to give a constant image. In other words, while in dynamic view stories can appear and disappear, in the static view the stories are a given constant.

Differences between Applet and Standalone viewer

1. The applet display will in practice be considerably smaller than the standalone viewer. Therefore, the applet is less detailed and some physical laws might need to be bend a bit.

4 Architecture

The architecture of AMBER is defined in terms of modules, whiteboards and the messages they use to communicate. Figure 4.1 shows the flow of messages between the modules and whiteboards.

In the Section 4.1 the modules are described and in Section 4.2 the messages connecting them are defined.

4.1 Modules

A complete AMBER system will comprise at least three modules running at the same time; there is a Crawler module, a selector module called Sieve and a display called ShowOff. The modules are separate executables with their own life-cycles and resources. Since TCP/IP is used, the executables don't need to be on the same machine to communicate.

Every module has a specified interface through which communication with Psyclone is handled.

4.1.1 Crawler modules

When the Crawler is started, it will create one of the available handlers (depending on what is specified on the command line or what is set as default during build time).

It also creates an AirBrush instance to communicate with Psyclone via JavaOpenAIR. The module name announced to Psyclone is 'Crawler.' plus the name of the handler, so 'Crawler.RSS' in case of the RSS handler.

After connecting with Psyclone, the handler can get its parameters stored in the psySpec file and go to work. It will post stories with type 'Story' on the whiteboard 'WB.Stories'.

RSS

The RSS crawler module will be fairly straightforward. There are actually quite a few good RSS parsers around for Java, the only thing the crawler should do is getting the stories from the RSS feeds along with meta-information and store it on the whiteboard.

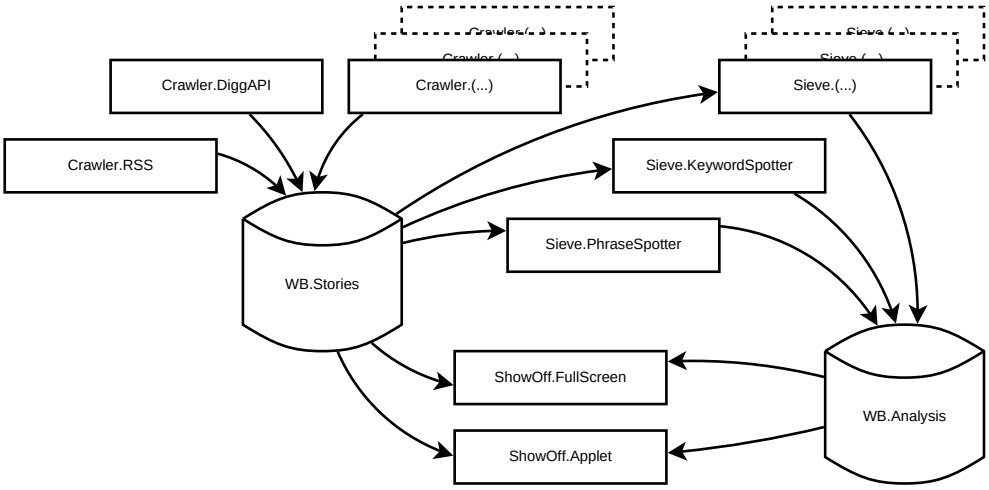

Figure 4.1: Global Amber architecture, the names are Psyclone module names

Although the module is called RSS, it can also handle Atom feeds, which is also quite a popular format.

Psyclone module specification for module Crawler.RSS:

Whiteboard	Type
Triggered by WB.Control	Feed.*
Post to WB.Stories	Story

DiggAPI

Digg is a website which lets users submit stories found on the web. Other users then moderate the submissions either by 'digging' or 'burying' a story. A story with a lot of 'diggs' is a popular one. The nice thing about Digg is that it actually does a lot of preprocessing work for the Amber system.

Digg announced[1] that they will publish a public API within the next months. If time allows, a DiggAPI module is created.

Psyclone module specification for module Crawler.DiggAPI:

Whiteboard	Type
Post to WB.Stories	Story

[1] `http://diggtheblog.blogspot.com/2006/07/digg-labs-launches-alpha.html`

4.1.2 Sieve modules

All analysis modules, or sieves, will get a trigger from a new story on the whiteboard WB.Stories. They analyse it and if there is anything to say about the story, a Analysis message is sent to the whiteboard WB.Analyses containing its judgement on the story.

The contents of this message is specified in Section 4.2.

Analysis modules may take any time they like to come to a verdict, but it is possible that a story has already disappeared from the visualization if the response is very late.

Since all modules employ the same external behaviour, the Psyclone specification is the same for every one of them.

Psyclone module specification for module Sieve.???:

Whiteboard	Type
Triggered by WB.Stories	Story
Post to WB.Analyses	Analysis

4.1.3 ShowOff modules

The ShowOff modules are visualizers which combine the crawled stories from the Crawler with the analyses from the Sieve modules.

Psyclone module specification for module ShowOff.???:

Whiteboard	Type
Triggered by WB.Stories	Story
Triggered by WB.Analyses	Analysis

Full screen

The full screen application will display a lot of information and is there to be looked - not glanced - at. It should be possible to let it do its job autonomously, just showing a pretty picture, or to be interactive.

Ambient applet

The ambient applet will display a very easy to understand image (a glance at it should be enough) of the status of the page it is on. I.e. if the page is a weblog, it should display subject information on that weblog, if it is on the page of a thread of a forum, it displays the flow of the discussion.

4.2 Messages

There are a few message types in the system. Two of which must be defined system-wide because they are used in the communication between modules.

4.2.1 Message type 'Story'

The Story message is only posted to the whiteboard 'WB.Stories' and only by Crawler modules.

The message content is a YAML Ain't Markup Language (YAML)[2] document which represents the storyData field inside the Java counterpart of the message. It contains at least the properties 'URI' (to identify the story, GUIDs are RSS specific and cannot be used), 'Author', 'Title', 'Story-Content'.

It may also contain 'Publication-Date' (which is the date of publication in Internet Message Format[RFC01, Section 3.3]), 'Kind' and other fields.

An example of a YAML document containing Story data:

```
---
URI: http://ijsland.luijten.org/2006/09/12/skyr-wasdanou/
Author: Christian Luijten
Title: Skyr... Wasdanou?
Publication-Date: Tue, 12 Sep 2006 21:03:54 +0000
Kind: weblog-posting
Story-Content: >
  Een van die dingen die bij een onbekende cultuur horen zijn de
  eetgewoonten. Elk land heeft zo z'n producten die je nergens
  anders kan krijgen. IJslands nationale zuivelproduct heet Skyr,
  elke oma kan het maken, al is het nogal een hoop werk. Daarom is
  het lange tijd (lees: gedurende de jachtige periode na de tweede
  wereldoorlog toen de Amerikanen hier de boel kwamen ophaasten)
  in ongebruik geraakt, maar op een gegeven moment kwam de vraag
  toch weer terug en zijn een aantal zuivelproducenten het
  industrieel gaan produceren.
```

4.2.2 Message type 'Analysis'

Analysis typed messages are posted on the 'WB.Analyses' whiteboard only by Sieve modules. They contain information about stories present on the 'WB.Stories' whiteboard.

[2] http://www.yaml.org/

The content of these messages is also YAML format. Story messages are coupled with Analysis messages through their 'URI' fields, so this must be present.

An example of a message issued by an analysis module checking for the topic 'Zuivel' (which means dairy products in Dutch):

```
---
URI: http://ijsland.luijten.org/2006/09/12/skyr-wasdanou/
Topic: Zuivel
Relation-Strength: 1.0
Author-Strength: 0.1
```

Its 'Relation-Strength' suggests high relevance of the content with the topic. However, the 'Author-Strength' suggests that the author isn't an authority in the field.

Every analysis module sends a message to the whiteboard if it thinks it is relevant. It is thus possible that the same URI will get multiple analysis results or nothing at all, the visualizer module must cope with this and merge the available information.

5 Detailed design

In this chapter, every single class is described in terms of public interface and functionality. Because of the fairly dynamic character of this project – new ideas come and go – this chapter will not be finished until the end of the project and will probably change regularly.

5.1 Files and directories

All source code will be in the directory `src/`. All classes are in the package `amber` or in a subpackage thereof. The Psyclone specification file `psySpec.xml` is found in `data/`. External libraries that are redistributed with AMBER are in `lib/`. The source of this document, the traineeship report and the website are located in `documentation/`.

The application is built using Apache Ant[1] and it can be imported into Eclipse[2]. It requires Java SDK version 1.5 or greater.

5.2 Classes

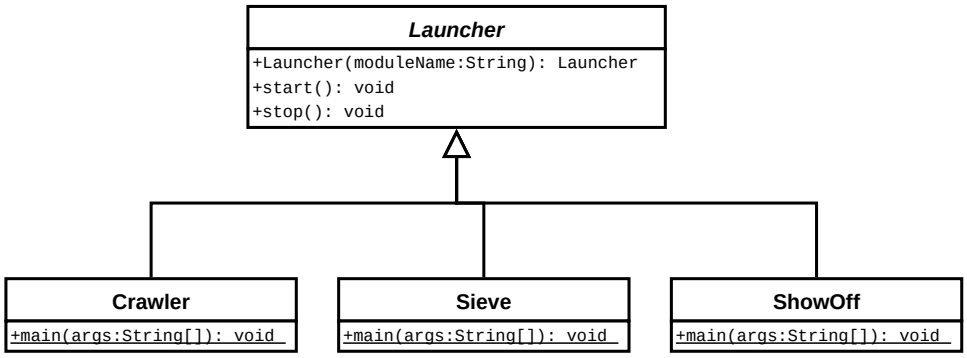

Figure 5.1: The inheritance model of the Launcher class

[1] `http://ant.apache.org/`
[2] `http://www.eclipse.org/`

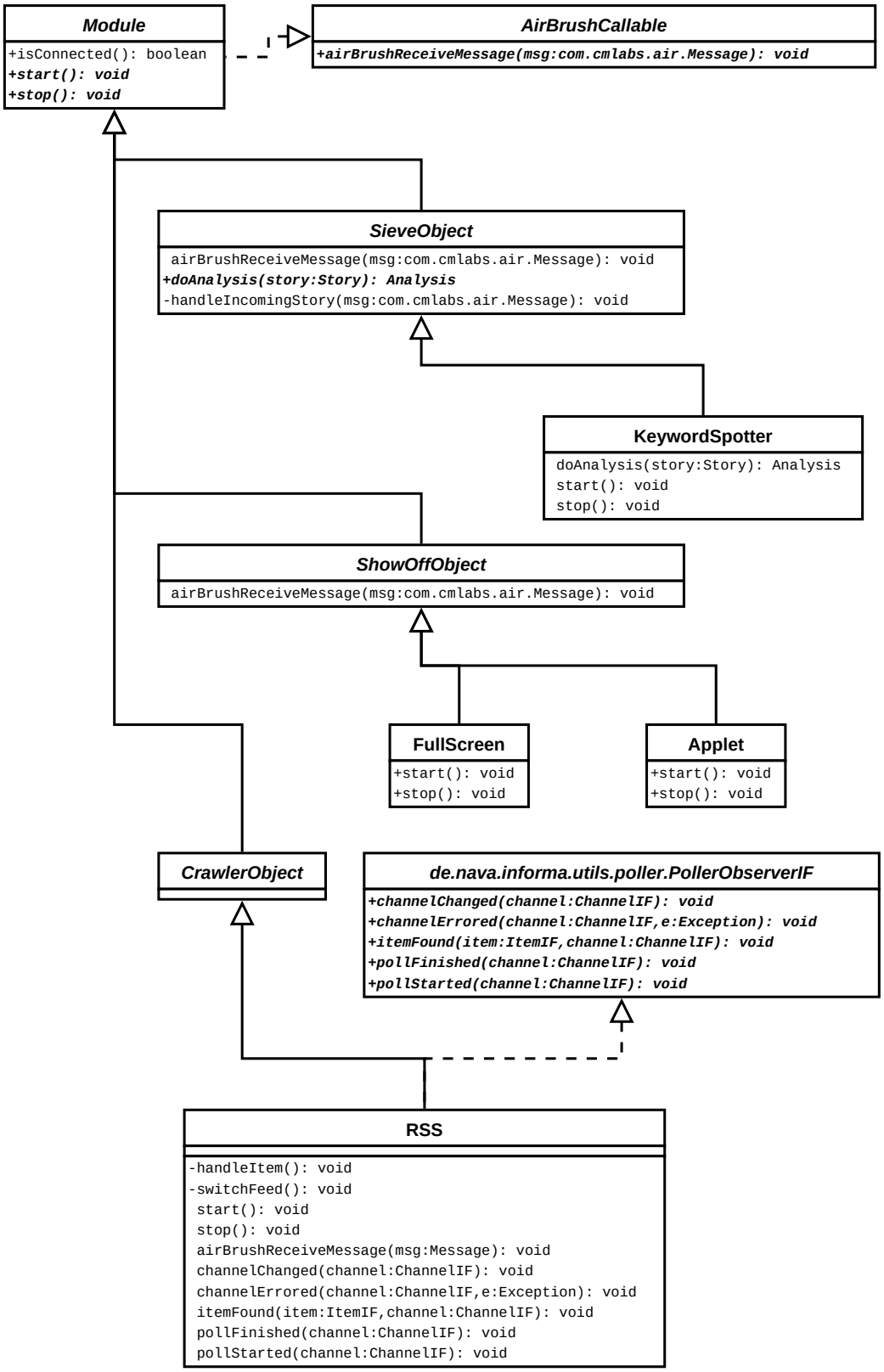

Figure 5.2: The inheritance model of the Module class

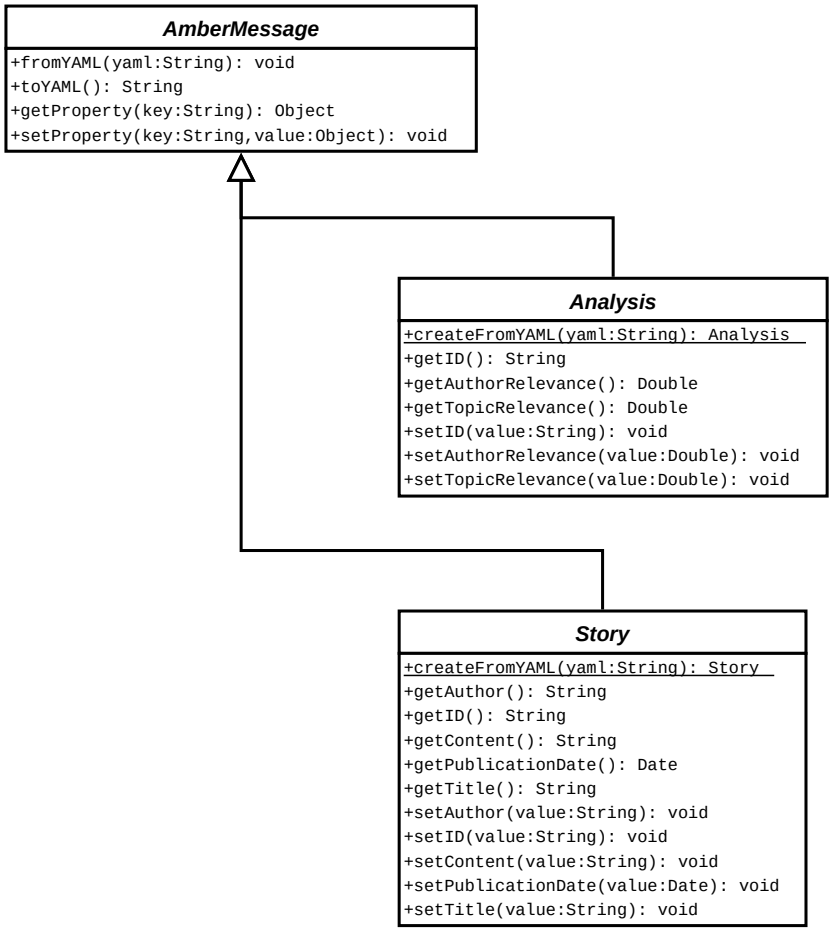

Figure 5.3: The inheritance model of the AmberMessage class

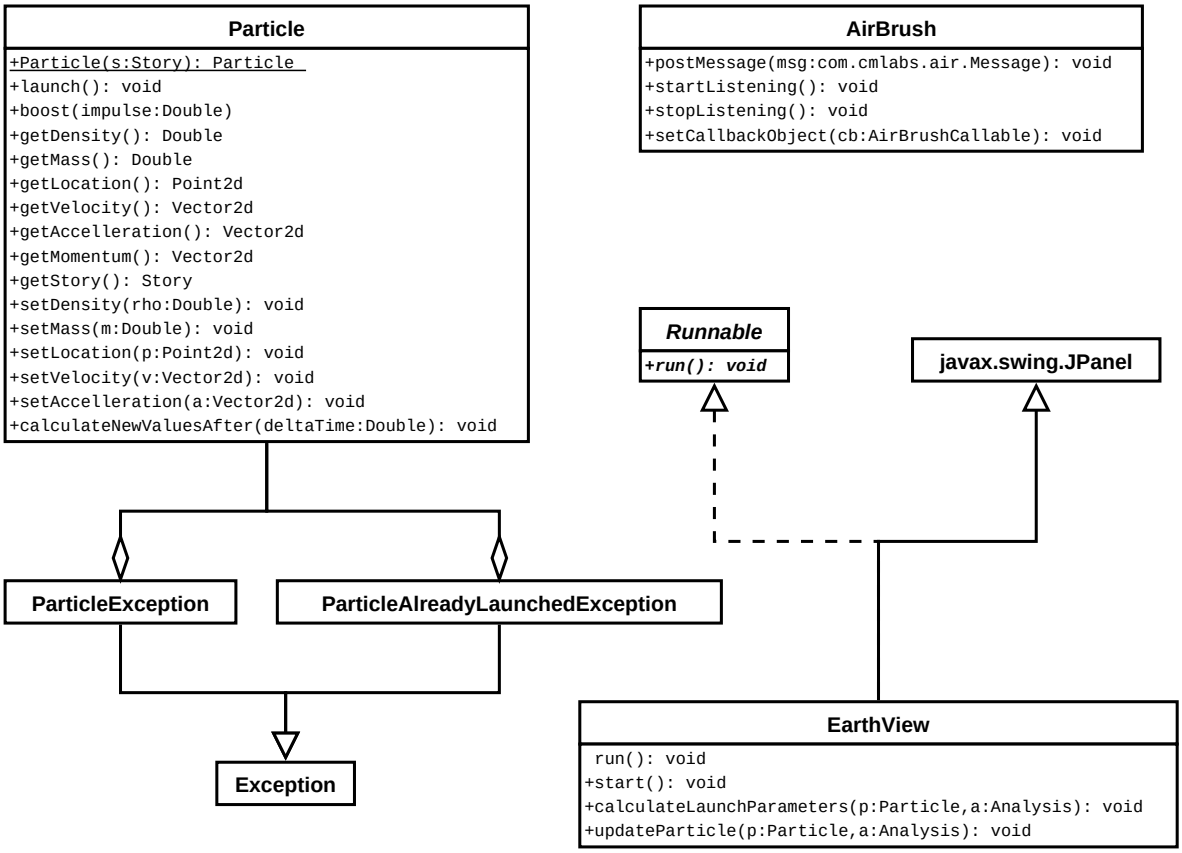

Figure 5.4: The inheritance model of the remaining classes

5.3 Objects in the amber package

The classes in the main package are launchers for the three different modules and abstract classes for the main objects of these modules. Basically it works like this; if the Crawler class is started it creates a CrawlerObject and starts it.

5.3.1 public class Crawler

Type: Class
Function: The Crawler object makes launching a crawler easy and uniform for all crawler types.

Interface

Returns	Name and argument	Description
new	Crawler()	Creates an instance of Crawler. Also creates a CrawlerObject and the connection to Psyclone.
static void	main(String args)	Method executed upon launch of the Crawler.
void	start()	Starts the Crawler.
void	stop()	Stops the Crawler.

5.3.2 public abstract class CrawlerObject

Type: Class
Function: The CrawlerObject class provides a uniform interface for crawler applications.
Implements: AirBrushCallable, ModuleInterface

5.3.3 public class ShowOff

Type: Class
Function: The ShowOff object makes launching a visualization module easy and uniform for all visualizer types.

5.3.4 public abstract class ShowOffObject

Type: Class
Function: The ShowOffObject class provides a uniform interface for visualization applications.
Implements: AirBrushCallable, ModuleInterface

Interface

Returns	Name and argument	Description
void	setStoryQueue(Queue⟨Story⟩ q)	Sets the queue of incoming stories. ShowOff will handle communication with the Psyclone whiteboard and puts stories in the queue.

5.3.5 public class Sieve

Type: Class
Function: The Sieve object makes launching a analysis module easy and uniform for all analyser types.

5.3.6 public abstract class SieveObject

Type: Class
Function: The SieveObject class provides a uniform interface for analysis applications.
Implements: AirBrushCallable, ModuleInterface

5.4 Objects in the amber.common package

5.4.1 public class AirBrush

Type: Class
Function: Ease communication with Psyclone through JavaOpenAIR.

Interface

Returns	Name and argument	Description
new	AirBrush(String moduleName, String hostName, int port)	Initializes a connection with Psyclone on *hostName:port* for module *moduleName*.
void	setCallbackObject(AirBrushCallable)	Sets the callback object to be used for calls from AirBrush.

5.4.2 public interface AirBrushCallable

Interface

Returns	Name and argument	Description
void	airBrushCallBack(com.cmlabs.air.Message msg)	Used as callback function for the AirBrush.

5.4.3 public interface ModuleInterface

Interface

Returns	Name and argument	Description
void	start()	Start the module.
void	stop()	Stop the module.

5.4.4 public class Story

Type: Class
Function: Storage of Story data.
Data: Story has a Map⟨String, Object⟩ field where it stores all data. It also holds a value with the number of analyses left until it is considered enough to be visualized. Instead of thrown back on the whiteboard with raw stories, it should then go the the processed stories.

Interface

Returns	Name and argument	Description
static void	createFromYAML(String yaml)	Creates a Story object, and initializes it from the *yaml* argument (see fromYAML method).
void	setAuthor(String author)	Sets the author of the story.
String	getAuthor()	Gets the author of the story.
void	setCreationTime(Date time)	Set the creation time of the story
Date	getCreationTime()	Get the creation time of the story.
void	setContent(String text)	Set the content of the story.
String	getContent()	Get the content of the story.
void	setValue(String k, Object v)	Store an arbitrary object v under keyword k.
Object	getValue(String k)	Gets the object stored under keyword k.
void	fromYAML(String yaml)	Parses a YAML string (e.g. coming from Psyclone) to the contents of the object (overwrites values currently stored!). Note, together with toYAML, this is the identity function: story.fromYAML(story.toYAML()) = story.
String	toYAML()	Converts the contents of the object to a YAML string to be sent via Psyclone.

5.5 Objects in the amber.crawler package

5.5.1 public class RSS

Type: Class
Extends: CrawlerObject
Implements: AirBrushCallable
Processing: An RSS object will accept messages to set its feed source, i.e. the feed it should crawl. A message of type 'Feed.RSS' or 'Feed.Atom' will set the source to that URL, whereas a message of type 'Feed.OMPL' will result RSS to parse this and set all URLs in the OPML document as source URL.
It will only post messages of the type 'Story' as they are described in Section 4.2.1. No other messages are sent.

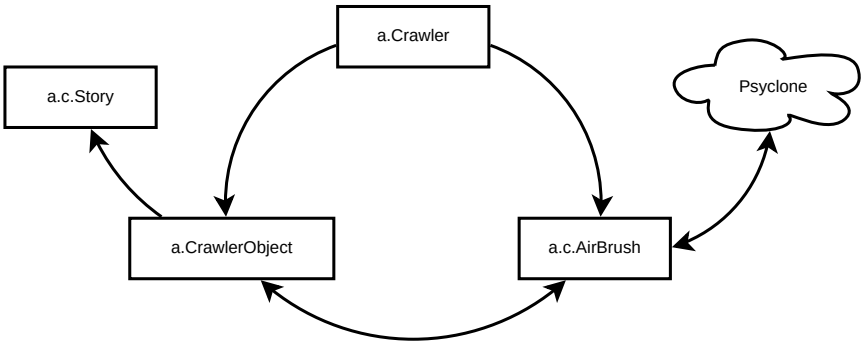

Figure 5.5: Diagram of the design of the Crawler, the names are Java classnames, arrows mean that calls exist in the direction of the arrow.

Interface

Returns	Name and argument	Description
new	RSS()	Creates a new RSS crawler. It will wait for instructions via Psyclone, like which RSS feed has to be monitored.
new	RSS(URL feedurl)	Creates a new RSS crawler, initialized with the URL of the feed to be monitored.
void	airBrushReceiveMessage-(Message msg)	Callback function for AirBrushCallable. It will handle incoming messages of the type 'Feed.RSS', 'Feed.Atom' and 'Feed.OPML'. With the OPML type it is possible to let a crawler handle more than one feed.
void	run()	Callback function for Runnable.

5.6 Objects in the amber.sieve package

5.6.1 public class KeywordSpotter

Type: Class
Extends: SieveObject
Implements: amber.common.AirBrushCallable
Function: The KeywordSpotter is configured to detect the presence of certain keywords in a story which makes it fit in a certain category or subject. In early versions the weights of the subjects will be equal. In later versions all weights of all subjects of a story must for instance add up to 100%. This gives the visualizer more freedom to place the story.

5.6.2 public class PhraseSpotter

Type:	Class
Extends:	SieveObject
Implements:	amber.common.AirBrushCallable
Function:	The PhraseSpotter will use a grammar engine to detect whether certain types of sentences appear in a story to classify it.

5.7 Objects in the amber.showoff package

5.7.1 public class EarthView

Type:	Class
Extends:	javax.swing.JPanel
Implements:	Runnable
Dependencies:	FullScreen, StoryLauncher
Function:	EarthView is a graphical user interface element based on a JPanel which displays a picture of Earth in space, with particles orbiting around it. Attractors can be added to influence the orbits of the particles.
Processing:	There is a list of particles, which contains information on their location and more. EarthView draws each of the particles in the list on itself.

Interface

Returns	Name and argument	Description
new	EarthView- (Collection⟨Particle⟩ pc)	Initializes the EarthView display. It is a child of JPanel and can as such be used inside any Swing application. Before starting the Earthview, first couple a Particle collection to it using setParticleCollection.
void	setParticleCollection- (Collection⟨Particle⟩ pc)	Sets the collection of particles to be displayed in the view.
void	run()	Runs the thread (for Runnable).
void	start()	Starts the thread, lets EarthView draw stuff.

5.7.2 public class FullScreen

Type:	Class
Extends:	amber.ShowOffObject
Implements:	amber.common.AirBrushCallable
Dependencies:	ShowOff
Function:	This is a ShowOff module which displays the visualization in full screen size.

Interface

Returns	Name and argument	Description
new	FullScreen()	Initializes something.
void	start()	Start the visualization.
void	airBrushReceiveMessage-(Message msg)	

5.7.3 public class Particle

Type:	Class
Function:	Storage of particle data.
Data:	An object of this class has information about its location and velocity, and it knows from which story it originates.
Dependencies:	StoryLauncher, Particles

Interface

Returns	Name and argument	Description
new	Particle(Story s)	Initializes a particle for Story s.
void	launch()	Launches the particle, all parameters must be set, they cannot be changed afterwards.
void	boost(double)	Boost the particle in the direction it is heading. This can happen when for instance the Story gets replies or comments; boosting keeps the particle around longer.
double	getMass()	Gets the mass of the particle
Point2d	getLocation()	Gets the location
Vector2d	getVelocity()	Gets the velocity
Vector2d	getAcceleration()	Gets the acceleration
void	setMass(double s)	Sets the mass of the particle to s
void	setLocation(Point2d v)	Sets the location to v
void	setVelocity(Vector2d v)	Sets the velocity to v
void	setAcceleration(Vector2d v)	Sets the acceleration to v
void	setNewValuesAfter(double t)	Calculates and sets new values using the current values after a period of time t

5.7.4 public class Applet

Type: Class
Extends: java.applet.Applet

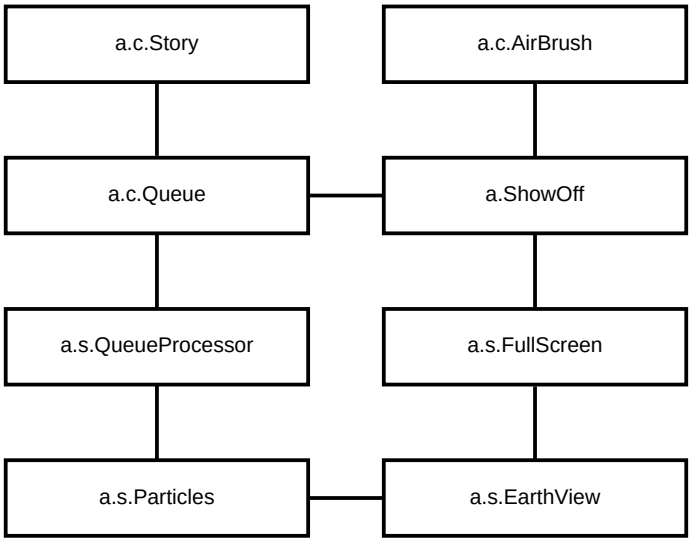

Figure 5.6: Diagram of the design of the full screen ShowOff module, the names are abbreviated Java classnames

Bibliography

[RFC01] *Rfc 2822 - internet message format*, 2001.

[Thó04] Kristinn R. Thórisson, Hrvoje Benko, Denis Abramov, Andrew Arnold, Sameer Maskey, and Aruchunan Vaseekaran. *Constructionist design methodology for interactive intelligences. AI Magazine*, 2004. 25(4):77–90.