Ambient Earth

Design Document — 1st draft

Christian Luijten
christian06@ru.is

November 8, 2006

Center for Analysis and Design of Intelligent Agents
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.

Christian Luijten (christian06@ru.is)

Contents

1 Introduction **5**
1.1 Quick introduction to Psyclone . . . 5

2 Usage scenarios **7**
2.1 A story is posted to a weblog . . . 7
2.2 A discussion is held on a web forum . . . 7

3 Requirements **8**
3.1 Extra-functional requirements . . . 8
3.2 Functional requirements . . . 8
3.2.1 Inputs . . . 8
3.2.2 Outputs . . . 9
3.2.3 Storage . . . 9
3.2.4 Computations . . . 9
3.2.5 Timing and synchronization . . . 9
3.2.6 Story analysis . . . 9
3.2.7 Visualization . . . 9

4 Architecture **11**
4.1 Modules . . . 11
4.1.1 Crawler modules . . . 11
4.1.2 Sieve modules . . . 12
4.1.3 ShowOff modules . . . 13
4.2 Messages . . . 14
4.2.1 Message type ‘Story’ . . . 14
4.2.2 Message type ‘Analysis’ . . . 14

5 Detailed design **16**
5.1 Files and directories . . . 16
5.2 Classes . . . 16
5.2.1 Module class hierarchy . . . 16
5.2.2 AmberMessage class hierarchy . . . 18
5.2.3 Other classes . . . 19
5.3 Detailed class descriptions . . . 20
5.3.1 Package amber . . . 20
5.3.2 Package amber.common . . . 23
5.3.3 Package amber.crawler . . . 36
5.3.4 Package amber.showoff . . . 39
5.3.5 Package amber.sieve . . . 52

5.4 Sequences 53
5.4.1 Life cycle of a Crawler module 53
5.4.2 Life cycle of a Sieve module 55
5.4.3 Life cycle of a ShowOff module 55
5.4.4 Life cycle of a Story object 55
5.4.5 Life cycle of an Analysis object 55
5.4.6 Life cycle of a Particle object 55

References **57**

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 needed, which is given in the next section. For more information about Psyclone, refer to the full documentation[CM].

1.1 Quick introduction to Psyclone

CMLabs, the creator of Psyclone says this about their product:

> Psyclone™ is a powerful platform for building modular, distributed systems. It is the middleware of choice in systems where complexity management or interactivity is key.

For this project, it is enough to know that there are "modules" and "whiteboards". Via a specification file the user can decide which types of messages will be coming from which modules to which whiteboards and more importantly, which types of messages on which whiteboard will trigger an event in which module. There are many more possibilities with the system, but this is all functionality AMBER is using.

There are two types of modules, internal and external. We are only interested in external modules right now.

Below is an example of the specification of a module, in this case of the Applet module.

```
<module name="Module.ShowOff.Applet.Anonymous">
 <executable />
 <description>
  This module gets stories from the processed whiteboard and uses
  the story's meta-data to display the stories in a Java applet.
 </description>
```

```
 <spec>
  <trigger from="WB.Stories" type="Story" />
  <trigger from="WB.Control" type="All.*" />
  <trigger from="WB.Analyses" type="Analysis.*" />
 </spec>
</module>
```

It defines that it is an external module (by the `executable` tag which can also contain more information on how to launch the module) and that it requests triggers from the whiteboards WB.Stories, WB.Control and WB.Analyses, but only if the type of the message is Story, All.* or Analysis.* respectively. * matches everything, so both All.Start and All.Stop will trigger the applet.

So now that the module is defined, we can start Psyclone and it will expect the module to be present. If it is, the messages are sent. If it isn't, the module is temporarily deactivated until it signs on and no messages are sent to the module.

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 and recent 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. Parts of the system must accept triggers from Psyclone whiteboards.
5. Sources must be Rich Site Summary (RSS) feeds, possibly aggregated via an Outline Processor Markup Language (OPML) file. The system should however be prepared to support other source types as well (i.e. it should be easily expandable).
6. 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. FIXME: The orbits follow Kepler's laws of planetary motion.
5. 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.

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, an analysis 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. It fetches the RSS feed from a set URL and produces a Story message for every new item that appears. The contents of the Story message is specified in Section 4.2.

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

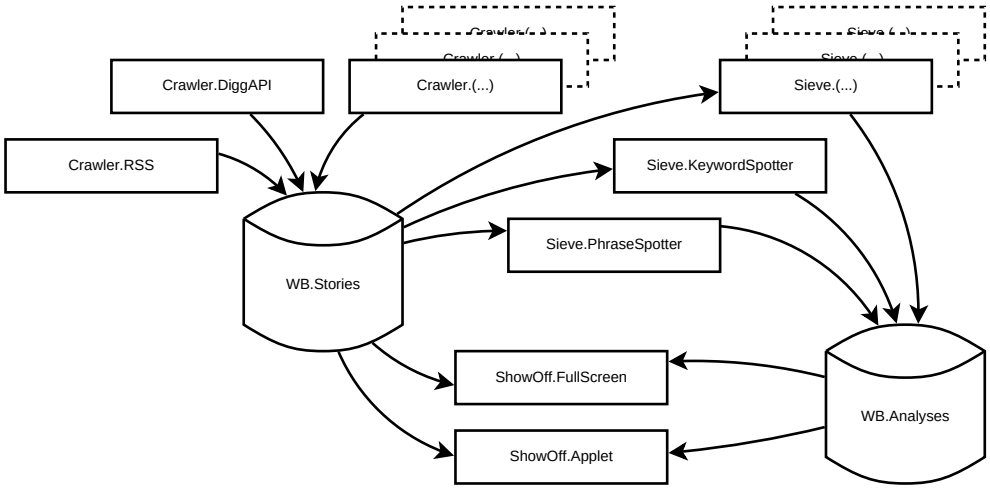

Figure 4.1: Global AMBER architecture, the names are Psyclone module names

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

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 it can say anything about the story, an 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.

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

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 regardless of their functionality 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 shows the particles orbiting the earth, being attracted to various attractors hanging around it, depending on their topics.

In future versions it is possible to get extra information about stories by hovering over the particles (i.e. it should not be impossible to implement this within the current architecture).

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[2].

The applet has no high priority, and it would be enough to convert the full screen application to an applet.

[2]Because currently there is no way to track discussions due to lack of meta-information about how comments relate to eachother, this part won't be implemented

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)[3] 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[Res01, 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.

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.

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

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 code object 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 written using Eclipse[1] and can be built using Apache Ant[2]. It requires Java SDK version 1.5 or greater.

To open the project in Eclipse, the following user libraries need to be defined: "Informa RSS Library" and "Jakarta Commons CLI". These are the only two depending libraries which aren't redistributed with AMBER the first is a redistribution of a collection of libaries already to be found at http://informa.sourceforge.net/, while the other is readily available at http://jakarta.apache.org/commons/cli/.

5.2 Classes

Since the project is written in Java, the code objects are classes. This section deals mainly with the relation between the class hierarchies and their function.

5.2.1 Module class hierarchy

The Module family is the collection of classes which can perform a production task within the AMBER system. They are the most important family of objects. The first distinction is the general functionality: crawling, analysing and visualising.

Secondly, as there can be more than one means to do one of the three main tasks, a class can extend one of these main modules. Thus we get the UML inheritance model as displayed in Figure 5.1.

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

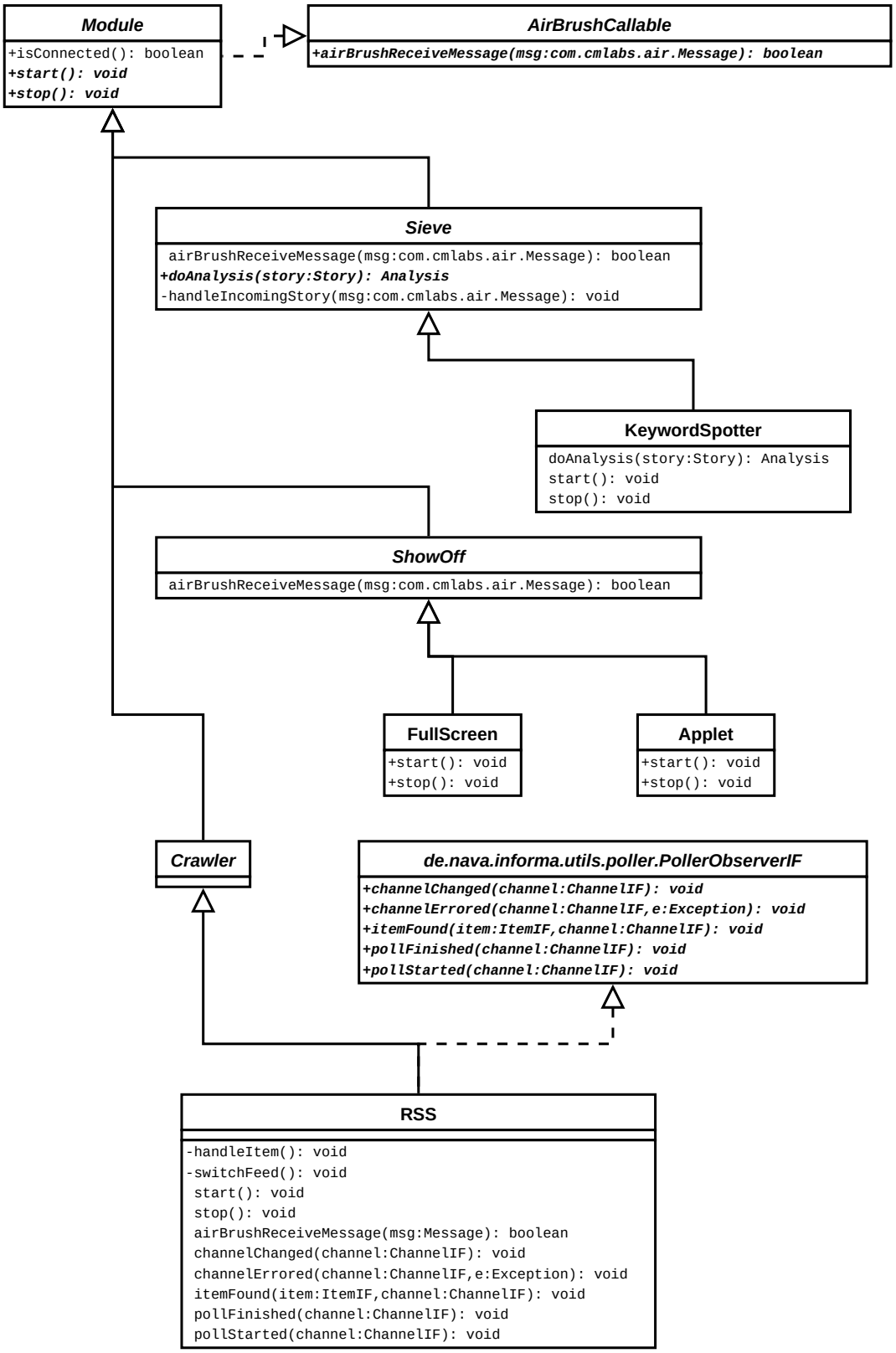

Figure 5.1: The inheritance model of the Module class

5.2.2 AmberMessage class hierarchy

AmberMessage objects are the holders of information sent and received via Psyclone. They provide means of (de)serializing data to and from Psyclone and in this way are a abstraction of the raw messages to the level of Java.

There are two subclasses of AmberMessage, Story and Analysis, which are to be used for posting on the WB.Stories and WB.Analyses whiteboards respectively.

A Module can introduce extra subclasses if it needs so for sending and receiving configuration data over Psyclone.

The UML inheritance model is shown in Figure 5.2.

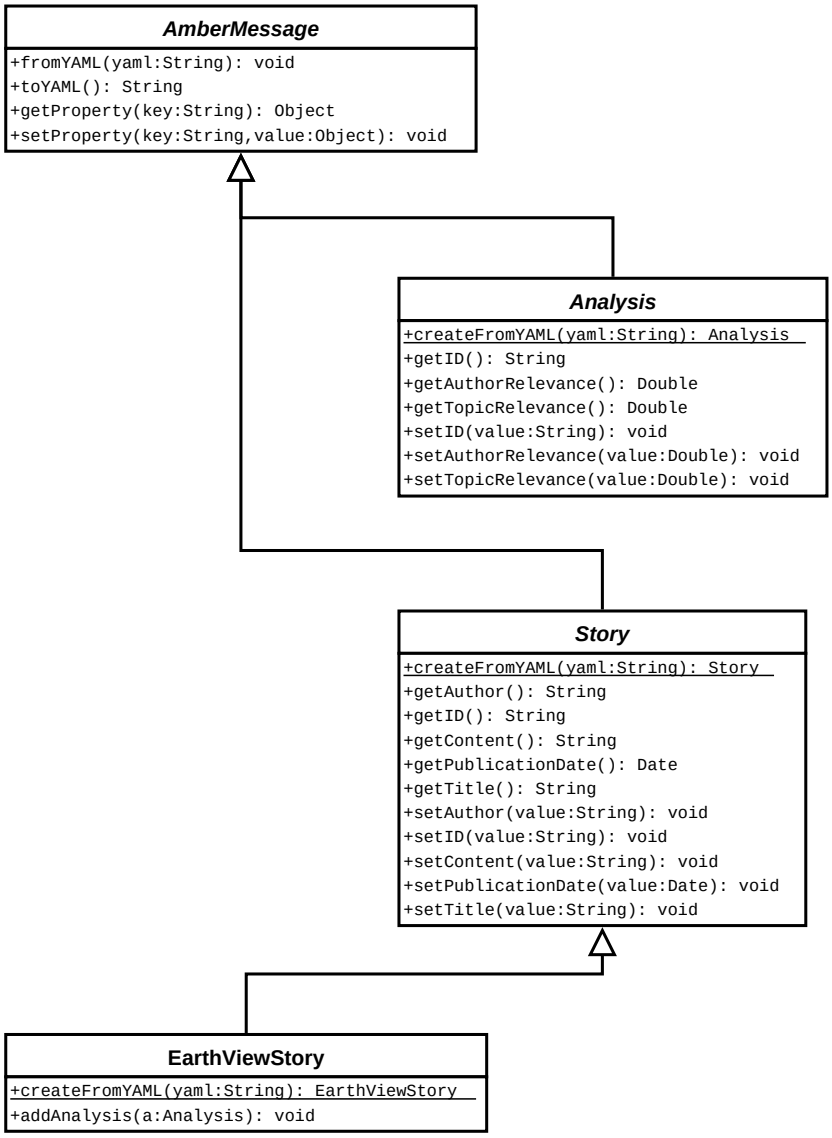

Figure 5.2: The inheritance model of the AmberMessage class

5.2.3 Other classes

There are classes which directly descend from the Java Object class and are as such outside the AMBER hierarchy. They are displayed in Figure 5.3.

The first is the Particle class, it represents a particle in the visualisation module. It is initialized to be "on the surface of the earth", then launch parameters are set according to the result of the first analysis which came in and it is launched. It will calculate its new position when the visualiser requests so.

Another class is the AirBrush class, which eases communication with the Java OpenAIR library.

The Launcher is a class which provides an interface to the command line and is responsible for starting and stopping the program. It has a command line argument parser which makes unwanted hard coding of parameters largely unneccesary.

Lastly, there is the EarthView class which is a Swing component to be placed in a window, which will draw Particles as they orbit around the earth.

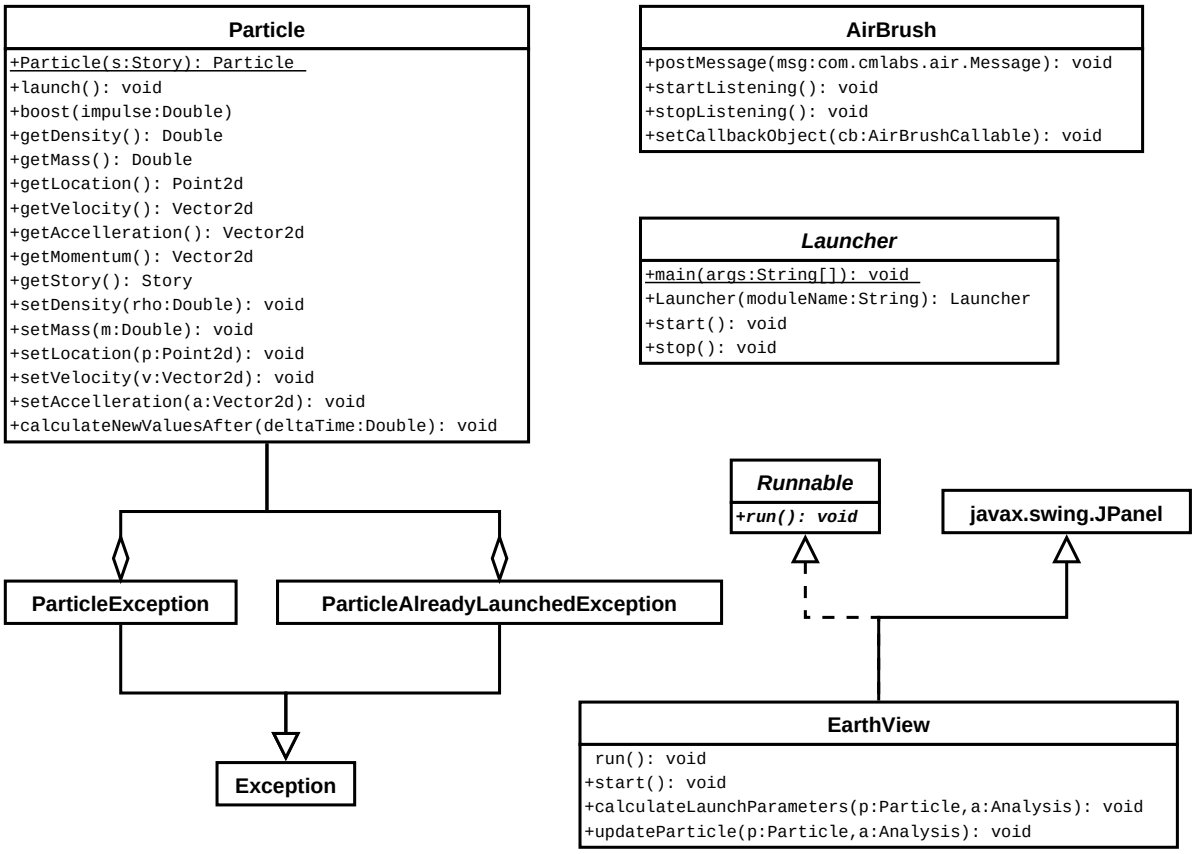

Figure 5.3: The inheritance model of the remaining classes

5.3 Detailed class descriptions

The next sections describe all classes in more detail, based on their location in the code base. The descriptions were automatically extracted from the sources via Javadoc and the TEXDoclet.

5.3.1 Package amber

Package Contents *Page*

Classes

Crawler 20
Gathers information, be it from the internet or from another information system.
ShowOff 21
Displays the information available in the whiteboards.
Sieve 22
Provides a means of adding meaning to a story.

Class Crawler

Gathers information, be it from the internet or from another information system. They post Story messages to a Psyclone whiteboard.

Declaration

public abstract class Crawler
extends amber.common.Module (in 5.3.2, page 31)

All known subclasses

RSS (in 5.3.3, page 36)

Constructors

- **Crawler**
  `public` **Crawler**`( java.lang.String` **moduleName**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Parameters**
    - `moduleName` – the name of the module to start
    - `hostname` – hostname of the psyclone server
    - `port` – port of the psyclone server

Class ShowOff

Displays the information available in the whiteboards. They combine Story messages with Analysis messages. It is dependent on the type of visualization one wants to accomplish what will be displayed and in what manner.

Declaration

public abstract class ShowOff
extends amber.common.Module (in 5.3.2, page 31)

All known subclasses

EarthViewWrapper (in 5.3.4, page 45), FullScreen (in 5.3.4, page 46)

Fields

- protected showoff.ObservableList **storyQueue**
  - In this list incoming stories are stored before they are handled.
- protected showoff.ObservableList **analysisQueue**
  - In this list incoming analyses are stored before they are handled.

Constructors

- **ShowOff**
  `public` **ShowOff**`( java.lang.String` **moduleName**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Parameters**
    - `moduleName` – the name of the module to start
    - `hostname` – hostname of the psyclone server
    - `port` – port of the psyclone server

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description copied from common.AirBrushCallable** (in 5.3.2, page 24)
    When a message comes in, this method is called on the set implementer of this interface.
  - **Parameters**
    - `msg` –
  - **Returns** – true if the message was handled, false if it wasn't

Class Sieve

Provides a means of adding meaning to a story. When a Story message comes in, the Sieve module will generate an Analysis based on it.

Declaration

public abstract class Sieve
extends amber.common.Module (in 5.3.2, page 31)

All known subclasses

KeywordSpotter (in 5.3.5, page 52)

Fields

- protected java.lang.String **topicString**
  - topic this Sieve creates Analyses for
- private final java.lang.String **messageTypeSuffix**
  - the suffix of messages to be sent to the whiteboard

Constructors

- **Sieve**
  `public` **Sieve**`( java.lang.String` **name**`, java.lang.String` **hostname**`, java.lang.In`
  port `)`
  - **Parameters**
    - `name` – the name of the module to start
    - `hostname` – hostname of the psyclone server
    - `port` – port of the psyclone server

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description copied from common.AirBrushCallable** (in 5.3.2, page 24)
    When a message comes in, this method is called on the set implementer of this interface.
  - **Parameters**
    - `msg` –

- **Returns** – true if the message was handled, false if it wasn't

- **doAnalysis**
  `protected abstract common.Analysis` **doAnalysis**`( common.Story` **story** `)`
  - **Parameters**
    - `story` – the story to be analysed
  - **Returns** – a newly created Analysis object which contains information about the story

- **handleIncomingStory**
  `protected void` **handleIncomingStory**`( com.cmlabs.air.Message` **msg** `)`
  - **Description**
    an incoming message gets analysed, if it is deemed relevant the analysis immediately sent on
  - **Parameters**
    - `msg` –

5.3.2 Package amber.common

Package Contents *Page*

Interfaces

AirBrushCallable 24
Specificies the callback functions for communication with AirBrush, the interface to JavaOpenAIR.

Classes

AirBrush 24
A small layer between JavaOpenAIR and Amber which holds state information about the connection with the Psyclone server and such.
AmberMessage 27
Superclass of all messages that are serialized and sent over to Psyclone: Story and Analysis.
Analysis 29
Holds information about a single Story object.
Launcher 30
To be called directly from the commandline and is in fact the preferred way to start any Amber module.
Module 31
Parent class of all Amber modules.
Polar2d 33
Represents polar coordinates: Angle and distance from origin.
Story 34
Holds all (meta-)information directly connected to a crawled story.

Interface AirBrushCallable

Specificies the callback functions for communication with AirBrush, the interface to JavaOpenAIR.

Declaration

public interface AirBrushCallable

All known subinterfaces

Sieve (in 5.3.1, page 22), ShowOff (in 5.3.1, page 21), Crawler (in 5.3.1, page 20), Module (in 5.3.2, page 31), RSS (in 5.3.3, page 36), EarthViewWrapper (in 5.3.4, page 45), FullScreen (in 5.3.4, page 46), KeywordSpotter (in 5.3.5, page 52)

All classes known to implement interface

Module (in 5.3.2, page 31)

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description**
    When a message comes in, this method is called on the set implementer of this interface.
  - **Parameters**
    * `msg` –
  - **Returns** – true if the message was handled, false if it wasn't

Class AirBrush

A small layer between JavaOpenAIR and Amber which holds state information about the connection with the Psyclone server and such.

Declaration

public class AirBrush
extends java.lang.Object
implements java.lang.Runnable

Fields

- private final com.cmlabs.air.JavaAIRPlug **plug**
  - the connection object
- private java.lang.Thread **thread**
  - thread handling incoming messages
- private AirBrushCallable **callback**
  - the callback object
- private java.lang.String **moduleName**
  - the string containing the name of the running module

Constructors

- **AirBrush**
  `public` **AirBrush**`( java.lang.String` **module**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Description**
    Creates and initializes AirBrush and connects to the Psyclone server.
  - **Parameters**
    - `module` – the name of the module to connect
    - `hostname` – hostname of the Psyclone server
    - `port` – port number of the Psyclone server

Methods

- **getParameterDouble**
  `public java.lang.Double` **getParameterDouble**`( java.lang.String` **key** `)`
  - **Description**
    Get the parameter with type Double
  - **Parameters**
    - `key` –
  - **Returns** – the parameter stored under key
- **getParameterInteger**
  `public java.lang.Integer` **getParameterInteger**`( java.lang.String` **key** `)`
  - **Description**
    Get the parameter with type Integer
  - **Parameters**
    - `key` –
  - **Returns** – the parameter stored under key

- **getParameterString**
  `public java.lang.String` **getParameterString**`( java.lang.String` **key** `)`
  - **Description**
    Get the parameter with type String
  - **Parameters**
    * `key` –
  - **Returns** – the parameter stored under key
- **hasParameter**
  `public boolean` **hasParameter**`( java.lang.String` **key** `)`
  - **Parameters**
    * `key` –
  - **Returns** – true when the parameter named key is present
- **openWhiteboard**
  `public boolean` **openWhiteboard**`( java.lang.String` **wb** `)`
  - **Description**
    Open connection a two-way connection with the whiteboard
  - **Parameters**
    * `wb` – name of the whiteboard to connect with
  - **Returns** – true if the whiteboard was succesfully opened
- **postMessage**
  `public void` **postMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description**
    Post a message to Psyclone
  - **Parameters**
    * `msg` –
- **run**
  `void` **run**`( )`
- **setCallbackObject**
  `public void` **setCallbackObject**`( AirBrushCallable` **cb** `)`
  - **Description**
    Set the callback object for handling incoming messages
  - **Parameters**
    * `cb` – the callback object
- **setParameter**
  `public void` **setParameter**`( java.lang.String` **key**`, java.lang.Double` **value** `)`
  - **Description**
    Set a parameter in Psyclone for the current module
  - **Parameters**
    * `key` –

        * value –

- **setParameter**
  `public void` **setParameter**`( java.lang.String` **key**`, java.lang.Integer` **value** `)`
  - **Description**
    Set a parameter in Psyclone for the current module
  - **Parameters**
    * key –
    * value –
- **setParameter**
  `public void` **setParameter**`( java.lang.String` **key**`, java.lang.String` **value** `)`
  - **Description**
    Set a parameter in Psyclone for the current module
  - **Parameters**
    * key –
    * value –
- **startListening**
  `public void` **startListening**`( )`
  - **Description**
    Start listening for incoming messages
- **stopListening**
  `public void` **stopListening**`( )`
  - **Description**
    Stop listening and disconnect from Psyclone

Class AmberMessage

Superclass of all messages that are serialized and sent over to Psyclone: Story and Analysis. Messages are serialized in YAML format.

Declaration

public abstract class AmberMessage
extends java.lang.Object

All known subclasses

Story (in 5.3.2, page 34), Analysis (in 5.3.2, page 29), EarthViewStory (in 5.3.4, page 44)

Fields

- private java.util.Hashtable **storage**
  - storage of the properties

Constructors

- **AmberMessage**
  `public` **AmberMessage**`( )`

Methods

- **fromYAML**
  `public void` **fromYAML**`( java.lang.String` **in** `)`
  - **Description**
    Converts a String generated by toYAML() to the correct contents of the object
  - **Parameters**
    - `in` –
- **getDoubleProperty**
  `public java.lang.Double` **getDoubleProperty**`( java.lang.String` **key** `)`
  - **Parameters**
    - `key` –
  - **Returns** – the value of the property stored under key
- **getProperty**
  `public java.lang.Object` **getProperty**`( java.lang.String` **key** `)`
  - **Parameters**
    - `key` –
  - **Returns** – the value of the property stored under key
- **getStringProperty**
  `public java.lang.String` **getStringProperty**`( java.lang.String` **key** `)`
  - **Parameters**
    - `key` –
  - **Returns** – the value of the property stored under key
- **setProperty**
  `public void` **setProperty**`( java.lang.String` **key**`, java.lang.Object` **value** `)`
  - **Parameters**
    - `key` –
    - `value` –
- **toYAML**
  `public java.lang.String` **toYAML**`( )`
  - **Returns** – the YAML representation of the contents of the Story object

Class Analysis

Holds information about a single Story object. Currently it is mainly focused on the relevance of a story within a certain topic, but it can be easily extended.

Declaration

public class Analysis
extends amber.common.AmberMessage (in 5.3.2, page 27)

Fields

- private static final java.lang.String **keyIdentifier**
- private static final java.lang.String **keyTopicRelevance**
- private static final java.lang.String **keyAuthorRelevance**
- private static final java.lang.String **keyTopic**
- private boolean **relevant**

Constructors

- **Analysis**
  `public` **Analysis**`( )`
- **Analysis**
  `public` **Analysis**`( java.lang.String` **identifier** `)`
  - **Parameters**
    - `identifier` – story identifier (for instance a URI)

Methods

- **createFromYAML**
  `public static Analysis` **createFromYAML**`( java.lang.String` **in** `)`
  - **Description**
    Creates a Analysis object from a YAML string. Used for instance when handling an incoming Analysis message to recreate the Analysis object.
  - **Parameters**
    - `in` – a YAML string representing the contents of an Analysis object
  - **Returns** – a newly created Analysis object, initialized with the contents of the YAML string
- **getAuthorRelevance**
  `public java.lang.Double` **getAuthorRelevance**`( )`

- **getID**
  `public java.lang.String` **getID**`( )`
- **getTopic**
  `public java.lang.String` **getTopic**`( )`
- **getTopicRelevance**
  `public java.lang.Double` **getTopicRelevance**`( )`
- **isRelevant**
  `public boolean` **isRelevant**`( )`
  - **Returns** – true if the analysis finds itself relevant
- **setAuthorRelevance**
  `public void` **setAuthorRelevance**`( java.lang.Double` **value** `)`
  - **Parameters**
    - `value` –
- **setID**
  `public void` **setID**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –
- **setRelevance**
  `protected void` **setRelevance**`( java.lang.String` **key**`, java.lang.Double` **value** `)`
  - **Description**
    Generalized version of the relevance setters
  - **Parameters**
    - `key` –
    - `value` –
- **setTopic**
  `public void` **setTopic**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –
- **setTopicRelevance**
  `public void` **setTopicRelevance**`( java.lang.Double` **value** `)`
  - **Parameters**
    - `value` –

Class Launcher

To be called directly from the commandline and is in fact the preferred way to start any Amber module. Start with -h command line switch to get help.

Declaration

public abstract class Launcher
extends java.lang.Object

Constructors

- **Launcher**
  `public` **Launcher**`( )`

Methods

- **main**
  `public static void` **main**`( java.lang.String[] args )`
  - **Description**
    Launch method
  - **Parameters**
    * `args` –
  - **Throws**
    * `java.lang.InstantiationException` –
    * `java.lang.IllegalAccessException` –
- **parseCommandLine**
  `private static org.apache.commons.cli.CommandLine` **parseCommandLine**`( java.lang.String[] args )`
  - **Parameters**
    * `args` –
  - **Returns** – an CommandLine object containing the parsed command line string
  - **Throws**
    * `org.apache.commons.cli.ParseException` –
- **printHelp**
  `private static void` **printHelp**`( org.apache.commons.cli.Options o )`
  - **Description**
    Prints help about the valid command line switches and options
  - **Parameters**
    * `o` – the options object

Class Module

Parent class of all Amber modules. Responsible for connectivity with Psyclone via AirBrush.

Declaration

public abstract class Module
extends java.lang.Object
implements AirBrushCallable

All known subclasses

Sieve (in 5.3.1, page 22), ShowOff (in 5.3.1, page 21), Crawler (in 5.3.1, page 20), RSS (in 5.3.3, page 36), EarthViewWrapper (in 5.3.4, page 45), FullScreen (in 5.3.4, page 46), KeywordSpotter (in 5.3.5, page 52)

Fields

- public final java.lang.String **moduleName**
- protected final AirBrush **airBrush**

Constructors

- **Module**
  `public` **Module**`( java.lang.String` **name**`, java.lang.String` **hostname**`, java.lang.` **port** `)`
  - **Parameters**
    - `name` –
    - `hostname` –
    - `port` –

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description copied from AirBrushCallable** (**in 5.3.2, page 24**)
    When a message comes in, this method is called on the set implementer of this interface.
  - **Parameters**
    - `msg` –
  - **Returns** – true if the message was handled, false if it wasn't
- **start**
  `public void` **start**`( )`
  - **Description**
    Start normal operation after initialization and configuration

- **stop**
  `public void stop( )`
  - **Description**
    Stop normal operation and exit

Class Polar2d

Represents polar coordinates: Angle and distance from origin. Roughly same function as Point2d and Vector2d, but without the calculations.

Declaration

public class Polar2d
extends java.lang.Object

Fields

- public double **theta**
  - value of the angle
- public double **r**
  - value of the distance from the origin

Constructors

- **Polar2d**
  `public Polar2d( )`
- **Polar2d**
  `public Polar2d( double[] v )`
  - **Parameters**
    - `v` – array of at least two elements.
- **Polar2d**
  `public Polar2d( double r, double theta )`
  - **Parameters**
    - `r` – distance from origin
    - `theta` – angle between the horizontal r axis extending to the right and the point

Methods

- **clone**
  `protected native java.lang.Object` **clone**`( ) throws java.lang.CloneNotSupportedEx`

- **fromCartesianTuple**
  `public static Polar2d` **fromCartesianTuple**`( javax.vecmath.Tuple2d t )`
  - **Description**
    Create polar coordinates out of a cartesian coordinates tuple.
  - **Parameters**
    * `t` –
  - **Returns** – the polar coordinates equal to the input tuple

- **scale**
  `public void` **scale**`( double f )`
  - **Description**
    Scale the distance from the origin
  - **Parameters**
    * `f` –

- **toCartesian**
  `private void` **toCartesian**`( javax.vecmath.Tuple2d t )`
  - **Description**
    Internal method to convert polar to cartesian coordinates. Because of difference between Point2d and Vector2d, this can't be done in one function.
  - **Parameters**
    * `t` –

- **toCartesianPoint**
  `public javax.vecmath.Point2d` **toCartesianPoint**`( )`
  - **Description**
    Returns the polar coordinates in Point2d cartesian coordinates.
  - **Returns** – the cartesian coordinates equivalent to the cartesian coordinates of the current object

- **toCartesianVector**
  `public javax.vecmath.Vector2d` **toCartesianVector**`( )`
  - **Returns** – the polar coordinates in Vector2d cartesian coordinates.

Class Story

Holds all (meta-)information directly connected to a crawled story.

Declaration

public class Story
extends amber.common.AmberMessage (in 5.3.2, page 27)

All known subclasses

EarthViewStory (in 5.3.4, page 44)

Fields

- private static final java.lang.String **keyIdentifier**
- private static final java.lang.String **keyAuthor**
- private static final java.lang.String **keyTitle**
- private static final java.lang.String **keyContent**
- private static final java.lang.String **keyPublicationDate**

Constructors

- **Story**
  `public` **Story**`( )`
- **Story**
  `public` **Story**`( java.lang.String` **identifier** `)`
  - **Parameters**
    - `identifier` –

Methods

- **createFromYAML**
  `public static Story` **createFromYAML**`( java.lang.String` **in** `)`
  - **Parameters**
    - `in` – a YAML string containing the contents of a Story object
  - **Returns** – a newly created Story object, initialized with the information in the YAML document
- **getAuthor**
  `public java.lang.String` **getAuthor**`( )`
- **getContent**
  `public java.lang.String` **getContent**`( )`
- **getID**
  `public java.lang.String` **getID**`( )`

- **getPublicationDate**
  `public java.util.Date` **getPublicationDate**`( )`
- **getTitle**
  `public java.lang.String` **getTitle**`( )`
- **setAuthor**
  `public void` **setAuthor**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –
- **setContent**
  `public void` **setContent**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –
- **setID**
  `public void` **setID**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –
- **setPublicationDate**
  `public void` **setPublicationDate**`( java.util.Date` **value** `)`
  - **Parameters**
    - `value` –
- **setTitle**
  `public void` **setTitle**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –

5.3.3 Package amber.crawler

Package Contents *Page*

Classes

RSS 36
Crawler module that can register with a RSS feed and will post Story objects whenever new stories come in.

Class RSS

Crawler module that can register with a RSS feed and will post Story objects whenever new stories come in. It also has a means of registering with more than one feed at a time, through the use of OPML.

Declaration

public class RSS
extends amber.Crawler (in 5.3.1, page 20)
implements de.nava.informa.utils.poller.PollerObserverIF

Fields

- private de.nava.informa.core.ChannelIF **channel**
- private de.nava.informa.utils.poller.Poller **poller**

Constructors

- **RSS**
  `public` **RSS**`( java.lang.String` **name**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Parameters**
    - `name` –
    - `hostname` –
    - `port` –
  - **Throws**
    - `java.net.MalformedURLException` –

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`
  - **Description copied from amber.common.AirBrushCallable** (in 5.3.2, page 24)
    When a message comes in, this method is called on the set implementer of this interface.
  - **Parameters**
    - `msg` –
  - **Returns** – true if the message was handled, false if it wasn't
- **channelChanged**
  `void` **channelChanged**`( de.nava.informa.core.ChannelIF` **arg0** `)`
- **channelErrored**
  `void` **channelErrored**`( de.nava.informa.core.ChannelIF` **arg0**`, java.lang.Exception` **arg1** `)`
- **convertToPrintable**
  `private java.lang.String` **convertToPrintable**`( java.lang.String` **input** `)`

  - **Description**

    Removes any unprintable characters. This is very rude (also removes all higher ASCII characters), but solves a lot of issues with serializing to YAML which isn't yet fully Unicode aware.
  - **Parameters**
    * `input` –
  - **Returns** – a safe string with only printable ASCII characters

- **getURL**
  `private java.net.URL` **getURL**`( ) throws java.net.MalformedURLException`

  - **Returns** – the URL as stored in the Psyclone parameter
  - **Throws**
    * `java.net.MalformedURLException` –

- **handleItem**
  `private void` **handleItem**`( de.nava.informa.core.ItemIF` **item** `)`
  - **Description**

    Creates a Story object with the contents of the item and posts it to Psyclone
  - **Parameters**
    * `item` –

- **itemFound**
  `void` **itemFound**`( de.nava.informa.core.ItemIF` **arg0**`, de.nava.informa.core.Channe`
  arg1 `)`

- **pollFinished**
  `void` **pollFinished**`( de.nava.informa.core.ChannelIF` **arg0** `)`

- **pollStarted**
  `void` **pollStarted**`( de.nava.informa.core.ChannelIF` **arg0** `)`

- **readAllItemsIn**
  `private void` **readAllItemsIn**`( de.nava.informa.core.ChannelIF` **c** `)`
  - **Description**

    Read all items in the channel and push them onto the whiteboard
  - **Parameters**
    * `c` –

- **registerChannel**
  `private void` **registerChannel**`( java.net.URL` **url** `)`
  - **Description**

    Parse, read all items and register the channel in the poller to be updated.
  - **Parameters**
    * `url` –

- **start**
  `public void` **start**`( )`

– **Description copied from amber.common.Module** (in 5.3.2, page 31)
Start normal operation after initialization and configuration

- **stop**
`public void stop( )`
  – **Description copied from amber.common.Module** (in 5.3.2, page 31)
  Stop normal operation and exit
- **switchFeed**
`private void switchFeed( )`
  – **Description**
  If the Psyclone parameter OPML exists, it will use that one for getting the feeds, otherwise it reads FeedURI
- **switchFeedOPML**
`private void switchFeedOPML( ) throws java.net.MalformedURLException`
  – **Description**
  Switch the all feeds for the ones in new OPML file
  – **Throws**
    * `java.net.MalformedURLException` –
- **switchFeedRSS**
`private void switchFeedRSS( ) throws java.net.MalformedURLException`
  – **Description**
  Switch the feed (only works when in RSS mode, not in OPML)
  – **Throws**
    * `java.net.MalformedURLException` –

5.3.4 Package amber.showoff

Package Contents *Page*

Classes

Applet 40
Implements an applet (!) containing the visualization module.
Attractor 40
Data object without functionality.
Demonstrator 41
Demonstrates the visual aspect of Amber.
EarthView 42
A Swing component displaying the main Amber visualization.
EarthViewStory 44
Holds both EarthView specific information like analyses and particles that belong to a story.
EarthViewWrapper 45

ShowOff module which creates the EarthView object and widgets to be used in standalone applications or applets alike.
FullScreen 46
Displays the EarthView component in a full screen window.
ObservableList 47
Enables the user to get a notification when there are changes in the list.
Particle 48
Representation of a Story in the EarthView visualization.
Particle.State 51

Class Applet

Implements an applet (!) containing the visualization module. Can thus be used to display on a website.

Declaration

public class Applet
extends javax.swing.JApplet

Fields

- private static final long **serialVersionUID**

Constructors

- **Applet**
  `public` **Applet**`( )`

Class Attractor

Data object without functionality.

Declaration

public class Attractor
extends java.lang.Object

Fields

- public amber.common.Polar2d **location**
  - The location of the attractor relative to the origin
- public java.lang.Double **force**
  - The force the attractor excites
- public java.lang.String **topic**
  - The topic the attractor represents

Constructors

- **Attractor**
  `public` **Attractor**`( )`

Class Demonstrator

Demonstrates the visual aspect of Amber. Generates stories and analyses about eight different countries and displays it. Implemented as an applet, so can be used on a website for demonstrations.

Declaration

public class Demonstrator
extends javax.swing.JApplet
implements java.lang.Runnable

Fields

- private static final long **serialVersionUID**
- static ObservableList **storyQueue**
- static ObservableList **analysisQueue**
- static java.lang.Thread **thread**
- int **storyCounter**

Constructors

- **Demonstrator**
  `public` **Demonstrator**`( )`

Methods

- **main**
  `public static void` **main**`( java.lang.String[] args )`
  - **Parameters**
    - `args` –
- **run**
  `void` **run**`( )`

Class EarthView

A Swing component displaying the main Amber visualization.

Declaration

public class EarthView
extends javax.swing.JPanel
implements java.lang.Runnable, java.util.Observer

Fields

- private static final long **serialVersionUID**
- java.awt.Graphics **offGraphics**
- java.awt.Image **previousImage**
- java.awt.Dimension **offDimension**
- private int **frameDelay**
- private java.lang.Thread **animator**
- private int **frame**
- static java.util.Hashtable **attractors**
- static java.util.Hashtable **stories**
- static java.util.Hashtable **particles**
- private ObservableList **storyQueue**
- private ObservableList **analysisQueue**
- private boolean **firstFrame**

Constructors

- **EarthView**
  `public` **EarthView**`( ObservableList` **sq**`, ObservableList` **aq** `)`
  - **Parameters**
    - `sq` –
    - `aq` –

Methods

- **addAttractor**
  `public Attractor` **addAttractor**`( amber.common.Polar2d` **location**`, java.lang.Double` **force**`, java.lang.String` **topic** `)`
  - **Description**
    Add an attractor
  - **Parameters**
    - `location` – the location of the attractor in polar coordinates
    - `force` – the force of the attractor
    - `topic` – the topic the attractor represents
  - **Returns** – the newly created attractor
- **drawParticle**
  `private void` **drawParticle**`( java.awt.Graphics` **g**`, Particle` **p** `)`
  - **Parameters**
    - `g` – the graphics object to be drawn upon
    - `p` – the particle object to be drawn
- **getNewAnalyses**
  `private void` **getNewAnalyses**`( )`
  - **Description**
    Get all new analyses from the analysis queue
- **getNewStories**
  `private void` **getNewStories**`( )`
  - **Description**
    Get all new stories from the story queue.
- **paintComponent**
  `protected void` **paintComponent**`( java.awt.Graphics` **arg0** `)`
- **run**
  `void` **run**`( )`
- **start**
  `public void` **start**`( )`
  - **Description**
    Start the animator

- **update**
  `void` **update**`( java.util.Observable` **arg0**`, java.lang.Object` **arg1** `)`

Class EarthViewStory

Holds both EarthView specific information like analyses and particles that belong to a story.

Declaration

public class EarthViewStory
extends amber.common.Story (in 5.3.2, page 34)

Fields

- private java.util.List **analyses**
- private java.util.Hashtable **weights**
- private Particle **particle**

Constructors

- **EarthViewStory**
  `public` **EarthViewStory**`( )`
- **EarthViewStory**
  `public` **EarthViewStory**`( java.lang.String` **identifier** `)`
  - **Parameters**
    - `identifier` –

Methods

- **addAnalysis**
  `public void` **addAnalysis**`( amber.common.Analysis` **a** `)`
  - **Parameters**
    - `a` –
- **calculateWeights**
  `public void` **calculateWeights**`( )`
- **createFromYAML**
  `public static EarthViewStory` **createFromYAML**`( java.lang.String` **in** `)`

  - **Parameters**

        * `in` – a YAML string representing the contents of a Story object
      - **Returns** – a newly created EarthViewStory, initialized with the information in the input

- **getWeight**
  `public java.lang.Double` **getWeight**`( java.lang.String` **topic** `)`
    - **Parameters**
      * `topic` –
    - **Returns** – the weight of the story

- **hasParticle**
  `public boolean` **hasParticle**`( )`
    - **Returns** – true if the story is associated with a particle

- **init**
  `private void` **init**`( )`

- **setParticle**
  `public void` **setParticle**`( Particle` **p** `)`
    - **Parameters**
      * `p` –

Class EarthViewWrapper

ShowOff module which creates the EarthView object and widgets to be used in standalone applications or applets alike. The field earthView contains the EarthView object which can be inserted into any graphical container.

Declaration

public class EarthViewWrapper
extends amber.ShowOff (in 5.3.1, page 21)

All known subclasses

FullScreen (in 5.3.4, page 46)

Fields

- public final EarthView **earthView**

Constructors

- **EarthViewWrapper**
  `public` **EarthViewWrapper**`( java.lang.String` **moduleName**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Parameters**
    * `moduleName` –
    * `hostname` –
    * `port` –

Methods

- **start**
  `public void` **start**`( )`
  - **Description copied from amber.common.Module** (in 5.3.2, page 31)
    Start normal operation after initialization and configuration

Class FullScreen

Displays the EarthView component in a full screen window.

Declaration

public class FullScreen
extends amber.showoff.EarthViewWrapper (in 5.3.4, page 45)

Fields

- private javax.swing.JPanel **list**

Constructors

- **FullScreen**
  `public` **FullScreen**`( java.lang.String` **moduleName**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  - **Parameters**
    * `moduleName` –
    * `hostname` –
    * `port` –

Methods

- **start**
  `public void start( )`
  - **Description copied from amber.common.Module** (in 5.3.2, page 31)
    Start normal operation after initialization and configuration

Class ObservableList

Enables the user to get a notification when there are changes in the list.

Declaration

public class ObservableList
extends java.util.Observable
implements java.util.List

Fields

- private java.util.List **list**

Constructors

- **ObservableList**
  `public ObservableList( )`
  - **Description**
    ObservableList is an implementation of List and extends Observable. On top of that, it is also synchronized.

Methods

- **add**
  `public void add( int arg0, java.lang.Object arg1 )`
- **add**
  `public boolean add( java.lang.Object arg0 )`
- **addAll**
  `boolean addAll( java.util.Collection arg0 )`
- **addAll**
  `boolean addAll( int arg0, java.util.Collection arg1 )`
- **clear**
  `void clear( )`

- **contains**
  `boolean` **contains**`( java.lang.Object` **arg0** `)`
- **containsAll**
  `boolean` **containsAll**`( java.util.Collection` **arg0** `)`
- **get**
  `java.lang.Object` **get**`( int` **arg0** `)`
- **indexOf**
  `int` **indexOf**`( java.lang.Object` **arg0** `)`
- **isEmpty**
  `boolean` **isEmpty**`( )`
- **iterator**
  `java.util.Iterator` **iterator**`( )`
- **lastIndexOf**
  `int` **lastIndexOf**`( java.lang.Object` **arg0** `)`
- **listIterator**
  `java.util.ListIterator` **listIterator**`( )`
- **listIterator**
  `java.util.ListIterator` **listIterator**`( int` **arg0** `)`
- **remove**
  `java.lang.Object` **remove**`( int` **arg0** `)`
- **remove**
  `boolean` **remove**`( java.lang.Object` **arg0** `)`
- **removeAll**
  `boolean` **removeAll**`( java.util.Collection` **arg0** `)`
- **retainAll**
  `boolean` **retainAll**`( java.util.Collection` **arg0** `)`
- **set**
  `public java.lang.Object` **set**`( int` **arg0**`, java.lang.Object` **arg1** `)`
- **size**
  `int` **size**`( )`
- **subList**
  `java.util.List` **subList**`( int` **arg0**`, int` **arg1** `)`
- **toArray**
  `java.lang.Object[]` **toArray**`( )`
- **toArray**
  `java.lang.Object[]` **toArray**`( java.lang.Object[]` **arg0** `)`

Class Particle

Representation of a Story in the EarthView visualization. When initialized, the particle will get some parameters to be launched. Upon launch, the particle will move according

to the equations of motion (v = v + a * t, s = 1/2 a * t∧2 etc.). If the story gets an analysis, the particle will be attracted to at least one of the attractors around the center. If it doesn't get an analysis, the particle will quickly fall down, crash and disappear. Relevant particles (i.e. the ones with analysis) will stay around for about 2 days (which is a constant to be chosen freely).

Declaration

public class Particle
extends java.lang.Object

Fields

- private amber.common.Polar2d **accel**
- private amber.common.Polar2d **velocity**
- private amber.common.Polar2d **location**
- private javax.vecmath.Point2d **locationCartesian**
- private Particle.State **state**
- private java.lang.Double **mass**
- private EarthViewStory **story**
- java.awt.Color **color**
- private boolean **bound**
- private static final java.lang.Double **CRASH_HEIGHT**
- private static final int **MAXIMUM_ORBITS_UNBOUND**
- private static final int **LIFE_LENGTH_IN_MS**
  - Particles of relevant stories should crash 48 hours after launch
- private java.util.Date **crashTime**

Constructors

- **Particle**
  `public` **Particle**`( EarthViewStory s )`
  - **Parameters**
    - `s` – the story which the particle should visualize

Methods

- **bind**
  `public void` **bind**`( )`

  - **Description**

    Indicate that a particle is bound to at least one analysis

- **calculate**
  `public void` **calculate**`( java.lang.Double` **time** `)`
  - **Parameters**
    - `time` –
- **crashed**
  `public boolean` **crashed**`( )`
  - **Returns** – true when the particle has crashed, false otherwise
- **displaceParticle**
  `private javax.vecmath.Point2d` **displaceParticle**`( )`
  - **Description**

    Particles get displaced by the attractors. This method calculates the exact displacement.
  - **Returns** – the cartesian coordinate point containing the actual location of a particle (after displacement by attractors)
- **getLocation**
  `public javax.vecmath.Point2d` **getLocation**`( )`
  - **Returns** – the cartesian location of the particle
- **getMass**
  `public java.lang.Double` **getMass**`( )`
- **getStory**
  `public amber.common.Story` **getStory**`( )`
- **getVelocity**
  `public javax.vecmath.Vector2d` **getVelocity**`( )`
  - **Returns** – the cartesian velocity of the particle
- **isLaunched**
  `public boolean` **isLaunched**`( )`
  - **Description**

    Method to check whether the particle has been launched
  - **Returns** – true if the particle has been launched
- **keplerRotation**
  `private java.lang.Double` **keplerRotation**`( java.lang.Double r )`
  - **Parameters**
    - `r` –
  - **Returns** – the velocity according to Kepler's second law ("A line joining a planet and its star sweeps out equal areas during equal intervals of time")
- **launch**
  `public void` **launch**`( )`

  - **Description**
    Launch the particle

- **setMass**
  `public void` **setMass**`( java.lang.Double` **m** `)`
  - **Parameters**
    * `m` –

- **setStory**
  `public void` **setStory**`( EarthViewStory` **s** `)`
  - **Parameters**
    * `s` –

Class Particle.State

Declaration

public static final class Particle.State
extends java.lang.Enum

Fields

- public static final Particle.State **NEW**
- public static final Particle.State **LAUNCH**
- public static final Particle.State **ORBITING**
- public static final Particle.State **CRASHING**
- public static final Particle.State **CRASHED**

Constructors

- **Particle.State**
  `private` **Particle.State**`( )`

Methods

- **valueOf**
  `public static Particle.State` **valueOf**`( java.lang.String` **name** `)`
- **values**
  `public static final Particle.State[]` **values**`( )`

5.3.5 Package amber.sieve

Package Contents *Page*

Classes

KeywordSpotter 52

Sieve module which matches a Story with a regular expression.

Class KeywordSpotter

Sieve module which matches a Story with a regular expression. If it matches, an Analysis object is created and a relevancy value is set and it is sent to the Psyclone whiteboard.

See also

– `java.util.regex.Pattern`

Declaration

public class KeywordSpotter
extends amber.Sieve (in 5.3.1, page 22)

Fields

- private java.util.regex.Pattern **contentPattern**
- private java.util.regex.Pattern **authorPattern**

Constructors

- **KeywordSpotter**
  `public` **KeywordSpotter**`( java.lang.String` **name**`, java.lang.String` **hostname**`, java.lang.Integer` **port** `)`
  – **Parameters**
    * `name` –
    * `hostname` –
    * `port` –

Methods

- **airBrushReceiveMessage**
  `boolean` **airBrushReceiveMessage**`( com.cmlabs.air.Message` **msg** `)`

– **Description copied from amber.common.AirBrushCallable** (in 5.3.2, page 24)

When a message comes in, this method is called on the set implementer of this interface.

– **Parameters**

* `msg` –

– **Returns** – true if the message was handled, false if it wasn't

- **doAnalysis**
`protected abstract amber.common.Analysis` **doAnalysis**`( amber.common.Story` **story** `)`

  – **Parameters**

    * `story` – the story to be analysed

  – **Returns** – a newly created Analysis object which contains information about the story

- **start**
`public void` **start**`( )`

  – **Description copied from amber.common.Module** (in 5.3.2, page 31)

  Start normal operation after initialization and configuration

- **stop**
`public void` **stop**`( )`

  – **Description copied from amber.common.Module** (in 5.3.2, page 31)

  Stop normal operation and exit

5.4 Sequences

This section describes the main courses of events.

The former three subsections describe the three types of main modules. They are all launched via the Launcher class which parses the command line and decides upon the command line switches which module will be loaded.

There are currently four modules: RSS, KeywordSpotter, FullScreen and Applet, which launch a Crawler, Sieve, ShowOff and ShowOff module, respectively.

The latter three subsections describe the life cycles of data elements in the system.

5.4.1 Life cycle of a Crawler module

Upon launch, Launcher parses the command line.

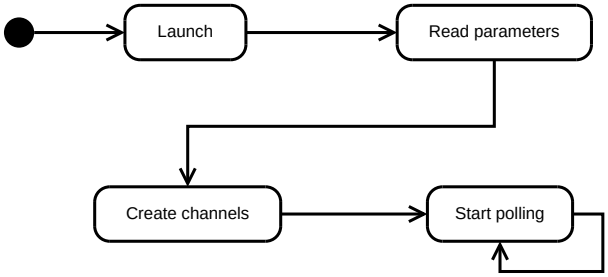

Figure 5.4: Sequence diagram of a Crawler module

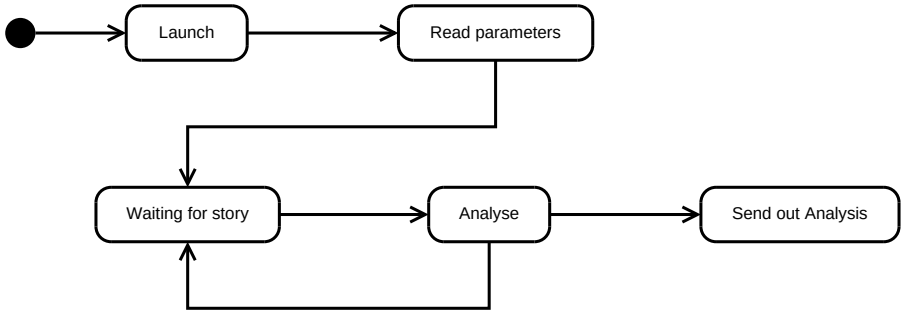

Figure 5.5: Sequence diagram of a Sieve module

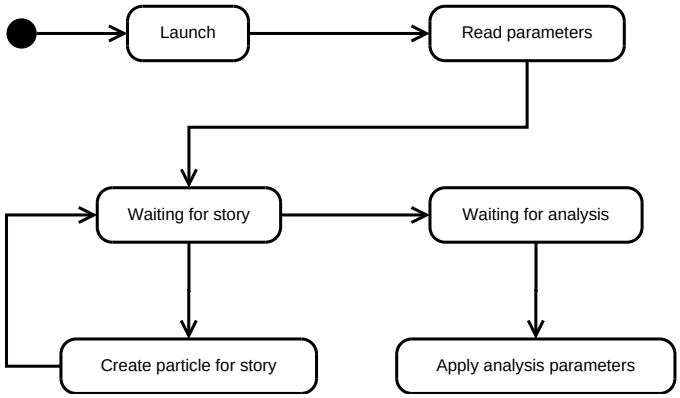

Figure 5.6: Sequence diagram of a ShowOff module

5.4.2 Life cycle of a Sieve module

5.4.3 Life cycle of a ShowOff module

5.4.4 Life cycle of a Story object

5.4.5 Life cycle of an Analysis object

5.4.6 Life cycle of a Particle object

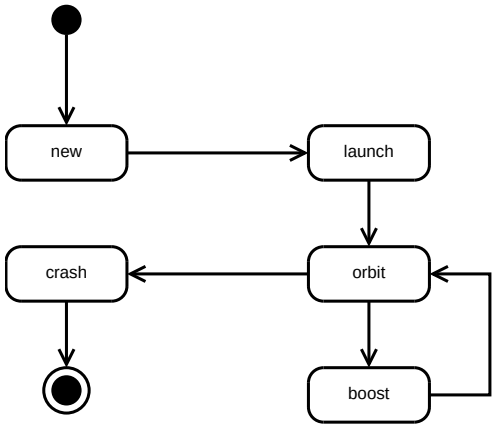

Figure 5.7: The life cycle model of a Particle object

Bibliography

[CM] Communicative Machines. *Psyclone manual*. http://www.cmlabs.com/psyclone/manual/.

[Res01] P. Resnick. *RFC 2822 - Internet Message Format*, 2001. ftp://ftp.rfc-editor.org/in-notes/rfc2822.txt.

[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.