CAVIAR

Context Aware Vision using Image-based Active Recognition

IST- 2001- 37540

Deliverable Report D 34

Software Report on Error Recovery

CAVIAR :	Software Report on Error Recovery
Date :	21 September 2005
Author(s) :	Rémi Emonet
Work package :	4
Document status :	Version 1.0
Usage :	public
Keywords :	Autonomic Software Architectures, Auto-regulation

Contents

Abstract

This deliverable document D34 describes the software for error recovery. Theoretic aspects and the details about the methods and motivations are to be found in Report D29. In the introduction, the reader will find a quick preview of the software structure and its interactions with the existing tracking system.

In section 2, the user interface of this software is described in detail, section 3 presents the software architecture. The technical documentation of the implementation generated with *Javadoc* can be found within the source code.

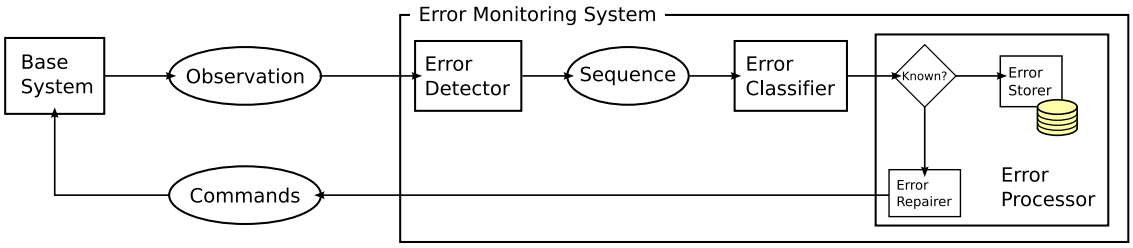

Figure 1: General architecture for error correction. In the implementation, the "Base System" is the tracking system, an "Observation" is a frame (the set of targets tracked at a given instant), and a "Sequence" is a sequence of frames over a time window.

1 Introduction

This document describes the piece of software developed for Error Recovery. Motivations, theoretical discussions and experimental results concerning the method are to be found into CAVIARreport on Error Recovery (D29). The developed application also allows the user to easily generate a scene reference model that is used by the error recovery monitor.

Globally the application implements the general architecture proposed in D29 (report on Error Recovery) which can be summed up by Figure 1. The error correction system monitors the flow of observations produced by the tracking system. An error detector is responsible for taking the decision that a sequence of observations is an error. An error sequence classifier then tries to attach a class label to the sequence. If the sequence corresponds to a known error class then the repair code associated with this class is called. If the error class is unknown, the sequence is stored in order to be classified latter.

This software program has been entirely developed in Java and monitors and controls a tracking system running in C++. The communication between the error recovery system and the tracking system are done using XML messages transfered through TCP connections.

Section 2 is a user manual describing how the software program is working and the various functionalities of the graphical user interface. Details about the software architecture can be found in section 3.

2 User Manual

This section gives indications about how to use the application and its graphical user interface (GUI). Section 2.3 details functionalites about acquiring the necessary scene reference model while section 2.4 explains how error examples are acquired and how they can be affected to existing or new error classes. Sections 2.1 and 2.2 give some basic concept to understand the sections presented previously.

2.1 Main databases into the application

The application is articulated around three main data sets.

- the scene reference model that is composed of a probability density function (an histogram in this implementation) and two sets of positive and negative examples. This scene reference model is used to associate a confidence factor to each of the target observed in the output of the tracking system the error recovery monitor is connected to;
- the error classes database that contains a set of error classes and associates to each error class a set of example error sequence for this error class and an error repair code among a set of possible repair codes;
- the "to classify" database which stores sequences that have been detected as error but can't have been affected to an existing error class.

2.2 Generalities

When the error recovery system is first started, it connects to a running tracker or wait for a tracker to start. The three datasets presented in the previous paragraph are directly manipulated by the program and can be loaded from and saved to files. At starting time, the error recovery system possibly loads some files containing the scene reference model, the error classes database and the "to classify" database (seldom used).

Once started, the application displays the main frame called "Manager" window (Figure 2). This frame is composed of a report area on the bottom and a set of controls on the top. The first messages displayed into the report area are the ones concerning the loading of the data files ("error" just meaning that no file has been found) and initialisation of the different modules of the system. Then comes the message concerning the connection attempt to the tracking system.

The "Manager" window allows the user to save the scene reference model and the error classes database using buttons (9). To lighten the GUI, the never-used button allowing the user to save the "to classify" database has been commented out in the source code. One can put this button back into the GUI if needed. The user can use button (8) to pop a new window up. This window shows in real time the output of the tracker: at any instant, an ellipse and an id for each target (Figure 3). To keep the cpu footprint of the error recovery monitor as low as possible, the current image is not displayed under the ellipses.

Using the "Manager" window, the user can also set the threshold that is used by the "Error Detector" to decide whether a sequence is a potential error. This threshold can be configured using (1).

2.3 Acquisition of the scene reference model

One of the functionality proposed by the application is to assist the user in the acquisition of a scene reference model. By the way, when the application is started for the first time

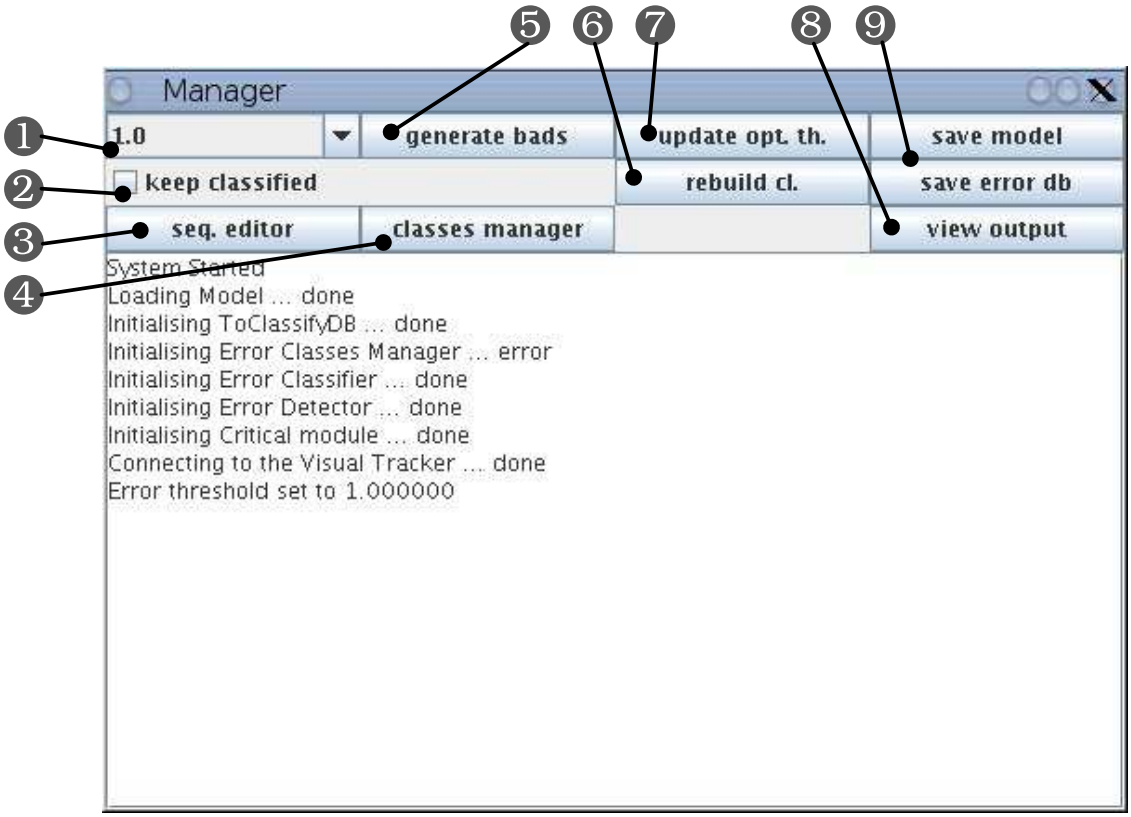

Figure 2: The Manager Window

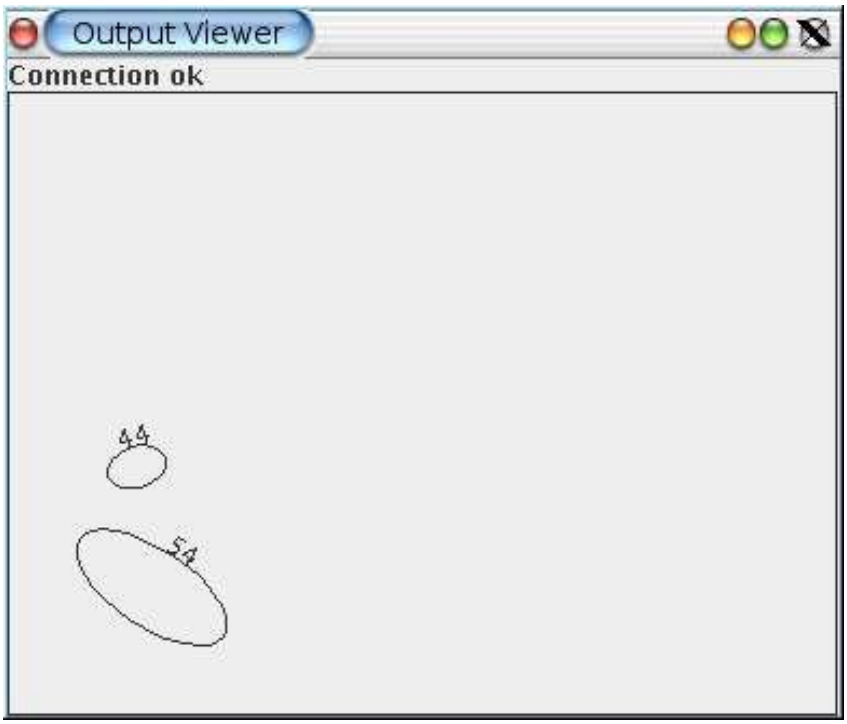

Figure 3: The Output Viewer showing two targets

for a given video camera, the scene reference model is completely empty unless we load a predefined scene model (that can for example be available from a ground truth labeling). The initial empty model systematically associate to each target a confidence factor of 0 and as a conclusion each observed sequence (a time window of the tracker output) will be considered as unknown and stored as an error sequence "to classify".

To process the stored "to classify" sequences, the button (3) of the "Manager" window pops up the "ToClassify Manager" window shown in Figure 4. This window is used both for the scene reference model acquisition and for the error classes creation and refinement. This window can be two modes ("waiting" or "editing") both shown in Figure 4.

The right panel (1) of the "ToClassify Manager" window is dedicated to the visualisation and edition of the current sequence (if any is picked). If no sequence is picked, the user can pick one (if any present in the database) using the button (2).

When in "editing" mode, the right panel (1) displays an extract (a subsequence) from the picked sequence. The user can control the starting frame of the extract and the number of frames in the extract using the keyboard: left and right arrows to choose the starting frame, up and down arrow to tune the length of the extract; shift and control modifier can also be used to navigate faster in the sequence. Dragging the mouse, the user can defined a rectangle in the right panel:

- if the user dragged the mouse with the right button then at each frame of the sequence being in the currently displayed extract, all the target having their center into the selection rectangle are removed from the sequence. This can be used when we want to remove some false targets outputted by the tracking system.

- if the user dragged the mouse with the right button then at each frame of the sequence being in the currently displayed extract, all the target having their center into the selection rectangle are merged. This can be used when we want to correct a false split from the tracking system.

The user can do various actions on the picked sequence. Some general ones are those triggered by buttons (4) and (5) which respectively drops the picked sequence and puts the sequence back into the "to classify" database. When an action is done on the picked sequence, the sequence is discarded unless the keep checkbox (3) is checked. When a sequence is discarded (an action is done and (3) is unchecked), the system picks another sequence (if available) if the auto pick checkbox (2) is checked else it switches to "waiting" mode. At any time, the user can use the clear button (C) to empty the "to classify" database. As picked sequences are not in the "to classify" database, the possibly picked sequence stays so.

To build the scene reference model, the user can use buttons (6) and (7). The scene reference model is composed of an histogram approximating the probability density function of what a normal tracker output is and two sets of positive and negative output examples. The two sets are used in the normalisation process described in D29. Button (6) adds all the targets from the picked sequence to the histogram while buttons (7) add all these targets into the set of positive (+) or negative examples (-). Use cases of the application

have shown that when the user wants to add some targets to the positive examples, he also wants to add them in the histogram: to optimise the usability of the GUI, the "Put In Model+" not only adds the targets of the sequence to the positive examples but also to the histogram.

The "Manager" frame also give some control on the scene reference model. Appart from saving it using button (9), the user can trigger the reevaluation of the optimal threshold of the model with button (7) (this optimal threshold is used in the normalisation process). As explained in D29 it can be difficult to obtain a lot of negative output examples and the user can generate some ones: when button (5) is clicked, the system uniformly generates some negative examples until the numbers of negative and positive examples become equal.

2.4 Acquisition of the error classes

The creation of the error classes can be done using the "ToClassify Manager" window. Using button (9) the user can create a new error class and affect the currently picked sequence to it. The user can also decide to add the currently picked sequence to any of the existing error classes using one of the buttons (8): there one such button for each existing class.

As the creation of the classifier can take some time, the classifier is not automatically rebuild at each addition of new example. The user can trigger the rebuild process from the "Manager" window using button (6). When the error detector detects an error, the error classifier tries to affect it to a known class. If it fails to do so, the sequence is stored in the "to classify" database; if it manages to classify, the repair code associated to the class is executed and the error sequence is discarded unless the checkbox (3) is checked. In such a case, the repair code is executed and the sequence is added to the "to classify" database.

The user can open the "Classes Manager" window using the button (4) from the "Manager" window. The "Classes Manager" window shown in Figure 5 is used to view and change the repair code associated to each error class. For each existing error class, a select allows the user to choose which repair codes (among the set of hand coded repair codes) is associated to the class and information about the class are given: the index of the class, its name and the number of examples that were collected for this class. On the bottom is also shown the number of "to classify" sequences on a button that pops the "ToClassify Manager" up.

3 Software Architecture

This section gives a general overview of the software architecture designed for the error recovery monitoring system. Full details of the developed classes can be found in the javadoc (generated from annotated sources).

The development of the application is done following a Model View Controller (MVC) pattern: the main principle is to minimise the coupling between the graphical user interface

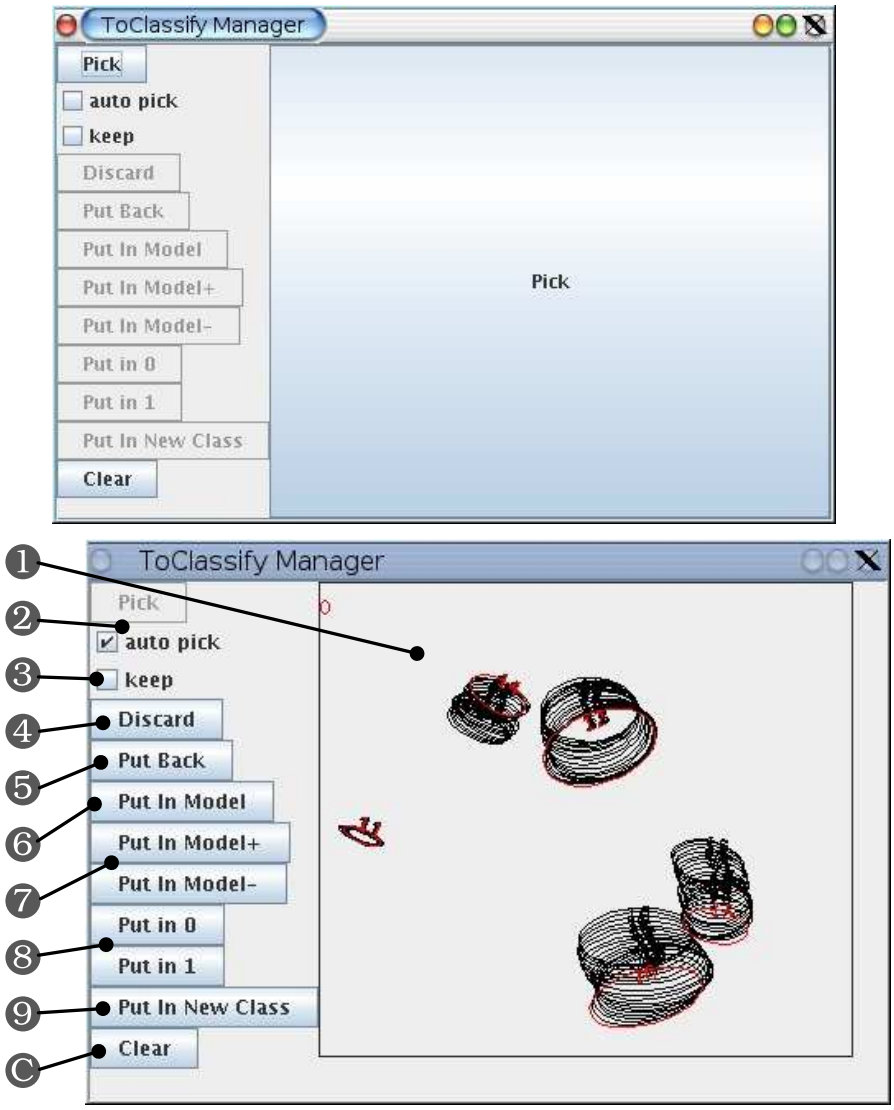

Figure 4: On top the "ToClassify Manager" in waiting mode: just waiting for the user to pick a sequence. On bottom the "ToClassify Manager" in editing mode: the sequence is visualised and can be modified.

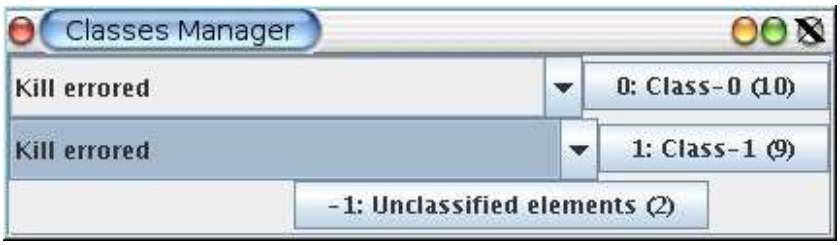

Figure 5: The Classes Manager Window with two classes

from the underlying model and in particular it removes all dependencies of the model to the GUI. Section 3.2 presents the model developed for the error recovery system.

3.1 View

The GUI is coded in Java Swing. The best documentation for a GUI is the user manual together with the source code. No particular widgets have been design for this application. The Independence of the model is mainly realised through the use of listener (Observer design pattern).

3.2 Model

A global and simplified overview of the package relations into the application can be found in Figure 6. Tool packages have been removed from this diagram to improve readability.

The first part of the model is the data structures presented in D29 that are used to represent the tracker output: TrackerTarget, TrackerFrame, TrackerSequence, TrackerHistory. Figure 8 shows an UML diagram with these classes. The PersistentObject interface is implemented by all the classes that can be stored or loaded from file.

The second part of the model is composed of all the class hierarchy representing the error recovery system by itself. As shown in Figure 7, a GlobalManager object is responsible for the communication between the different modules taking part in the error recovery process: it feeds the ErrorDetector with the tracker output, then if an error is detected it uses the ErrorClassifier to assign a class label to the error sequence and then uses the ErrorClassesManager to do the appropriate action depending on the error label affected to the sequence. These modules functionalities and requirement are described through interfaces. All these interfaces are generic (parametrised by one or more types) but the GlobalManager is concrete and uses instantiations of these generic interfaces. Each module interface is implemented in some concrete class as shown in Figures 9

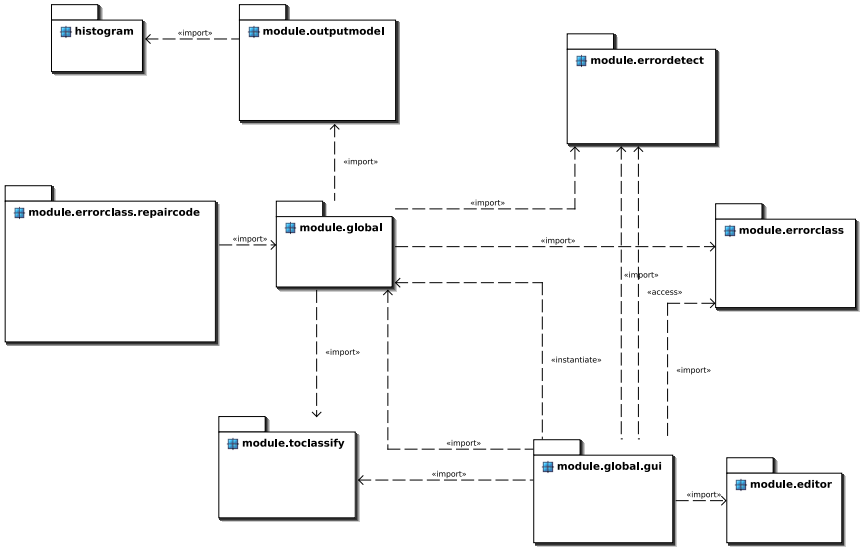

Figure 6: Overview of the packages

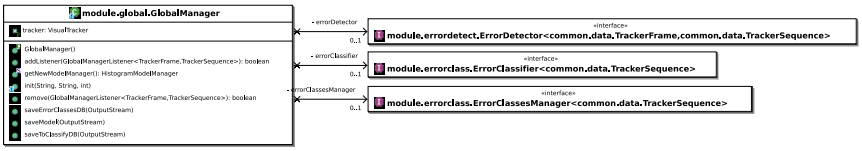

Figure 7: The GlobalManager and its managed components

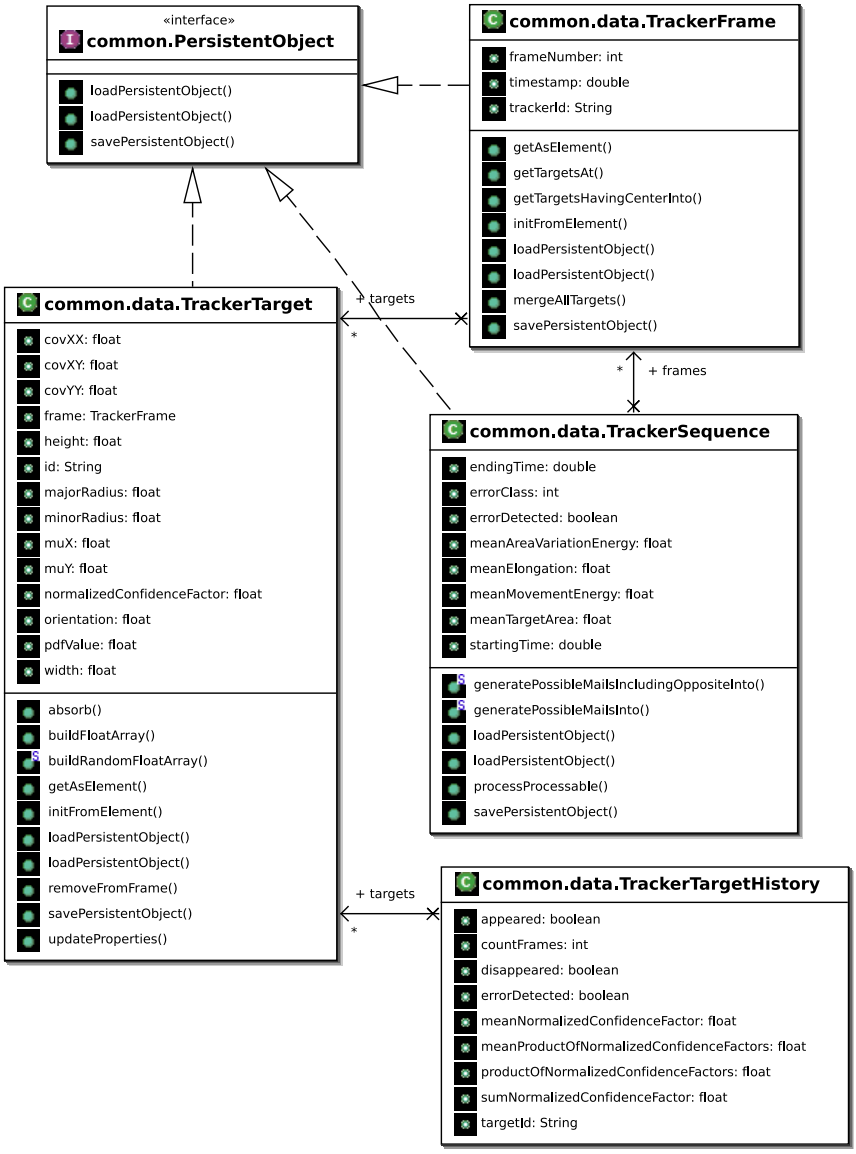

Figure 8: Data structures representing the tracking system output

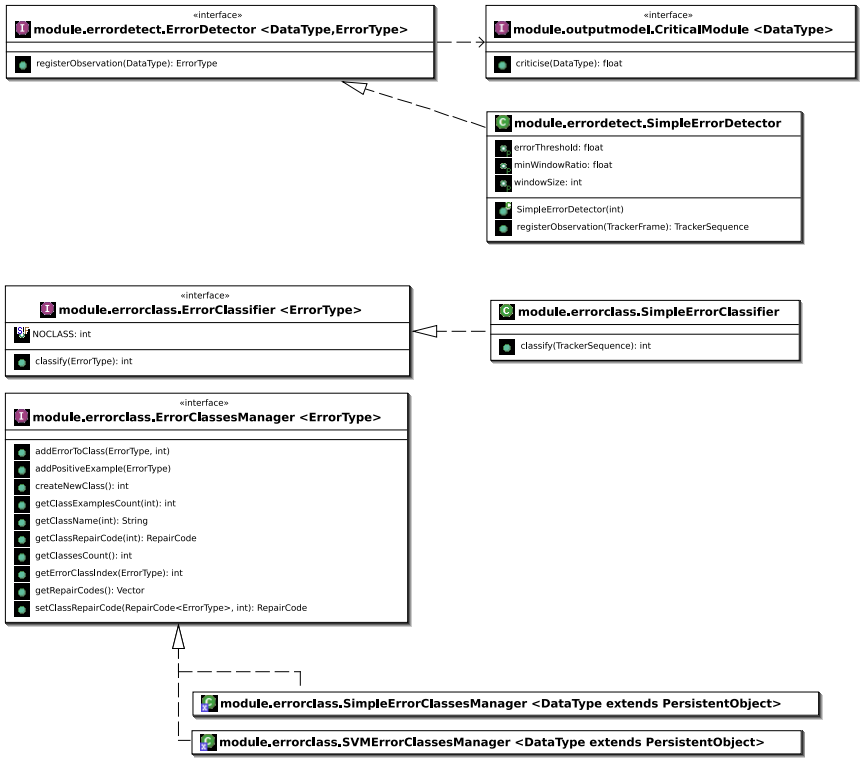

Figure 9: Implementations of the interfaces used by the GlobalManager