The Core Components of MathMoon

Christian von Schultz

20th December 2004

The game is composed of several "rooms", each with a mathematical task for the player to solve. The rooms are specified in a room list and thus configurable at run-time. The window contents changes depending on which room is currently loaded. If a room needs to act on some data a previous room has created, the data can be stored and retrieved from the application object, which in turn stores the data in a special `RGDataControl` object (see figure 1).

There are guard objects that wait for some event to occur. These come in two flavours: time guards and room guards. A time guard is activated every minute and can do everything from displaying a message reminding the user that there is only a limited amount of time left, to changing the room or even end the program because the user has run out of time (that's what the `TimeOut` time guard does, see figure 1). The `Timer` object is in charge of telling others when to wake up, and also tells the user (by sending messages to the `MainWindow`) how much time is left.

A room guard is activated each time we enter a new room, and may change certain aspects of that room by editing the `wxExpr` clause that will create the new room (each item in the room list is stored in a `wxExpr` object and written to the room list file as a Prolog clause, see figure 1 and the document describing the room list). A guard does not *have* to do anything when it is activated, it could also be used as a way of storing persistent data (that's what the `RGDataControl` room guard does).

The guards and the `WindowContents` typically reside in modules. A module is a DLL (under Windows) or a shared library (under GNU/Linux and UNIX) that contains a class derived from the abstract class defining the module type (i.e. `TimeGuard`, `RoomGuard` or `WindowContents`). It also contains a function that creates new instances of the class being defined in the module, and one function to delete these instances. Only modules having the same version as the main program can be used.

The modules are represented by the `Module` template class. It contains

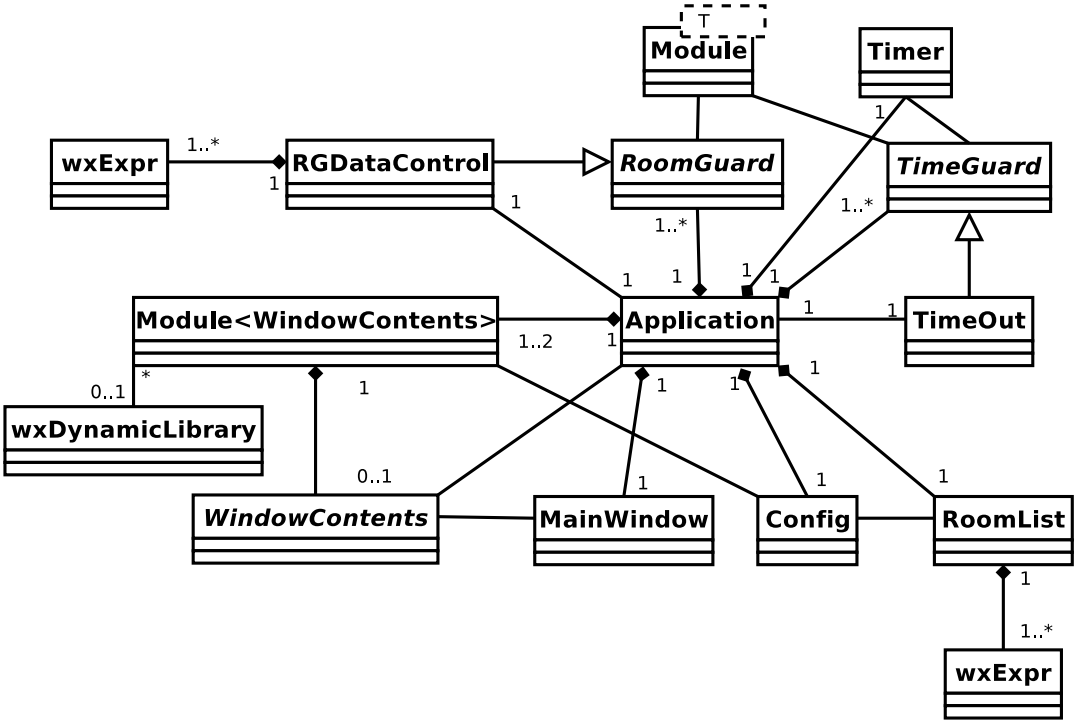

Figure 1: The core classes in MathMoon

functions for loading a DLL or shared library, verifying its version and loading the object it contains.

If an object needs to find some file, it consults the `Config` class. It uses command line parameters and the `wxConfig` object to determine different settings and file paths.

The `MainWindow` handles the menu bar and provides space for the `Timer` and `WindowContents` objects to interact with the user (the `Timer` object only updates the right pane of the status bar). The `MainWindow` also takes care of replacing the current `WindowContents`, when it is safe to do so.

The `WindowContents` object is responsible for (almost) all interaction with the user. When it is time for it to take over, `Init()` is called with the `wxExpr` object that is creating it and the `wxPanel` that the `MainWindow` has set up for usage by this object as arguments. This function may interact with the current `WindowContents` object before calling `Destroy()` on it and start populating the `wxPanel` itself. All `WindowContents` object should also support a `cHibernate()` function[1] that will create a `wxExpr` that should be able to recreate the object and its current state, if the user wants to save the session.

The `WindowContents` classes will in general be front-ends. They can act as templates for new rooms if they act differently depending on what the `wxExpr` clause that is creating them looks like. More information on the nature of these clauses can be found in the document describing the room list file.

At this point I guess I could describe the details of how all these objects communicate, but such things are subject to change. The best way of finding out exactly what the different parts do is probably to read the source code. If you wish to write a module for example, see how the existing modules were written. That will help you more than an in-depth description here. This document should still be useful as an overview though.

[1] All functions starting with a lower-case `c` create something that they do not delete. Deletion should be taken care of by the calling function or object.