Matrex Internal Structure

Table of Contents

Introduction

The purpose of this document is to give a correct vision of the internal architecture of Matrex, for the following purposes:

- for any user to understand how the system work; it can be useful for example when the system appears like **not behaving correctly**
- for any user to work with the **client/server** architecture that will be available in version 2.0
- as a start for developers to use the **API module directly**, without the GUI

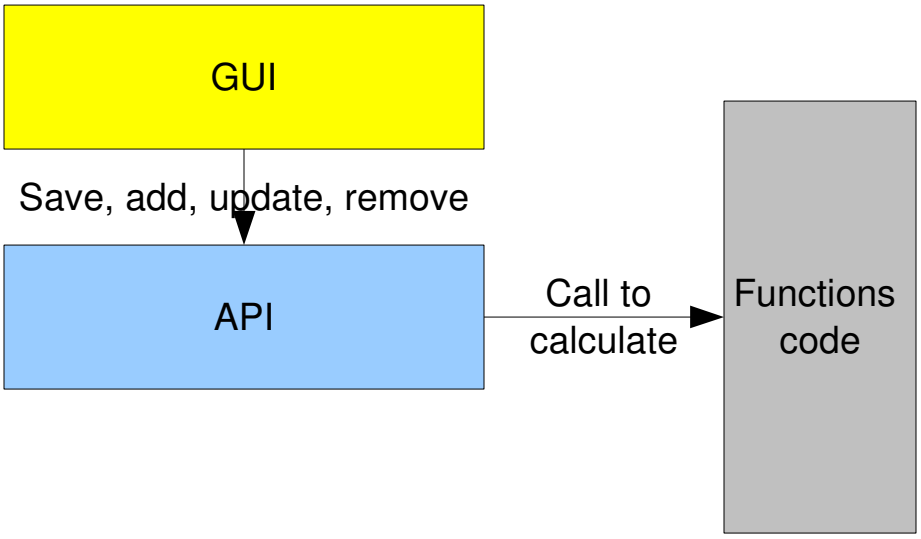
Since it serves these different purposes the document is technical, but not very detailed (it does not show code, for example) .

For who wants to know more, the Matrex javadoc is available.

Global structure

Matrex is composed internally by 3 modules:

- The **API**, i.e. the *matrex_api.jar* library, containing the classes in the *matrex.item* package. It is the kernel, the engine of Matrex. It is totally independent by the other two (it can potentially work alone).
- The **Functions**, i.e. the *matrex_fun.jar* library, containing the classes in the *matrex.fun* package, which implement the IFunction interface. Each class contains the code that implements a certain function template (for example the code to calculate *exp*, *inverse* or *variance)*.
- The **GUI**, i.e. the *matrex_gui.jar* library, containing the classes in the *matrex.gui* package. It is the graphical interfac for the API module.



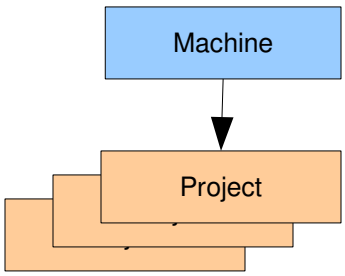
The API module

The following pictures show the business objects of the API module and their relationships.

Lets speak about **machines**. A machine represents a process that runs Matrex projects.

In version 1.0 there is only one machine, the Matrex desktop itself. From version 2.0 each Matrex desktop will be able to connect to other Matrex servers, and therefore to work with several machines in the same time.

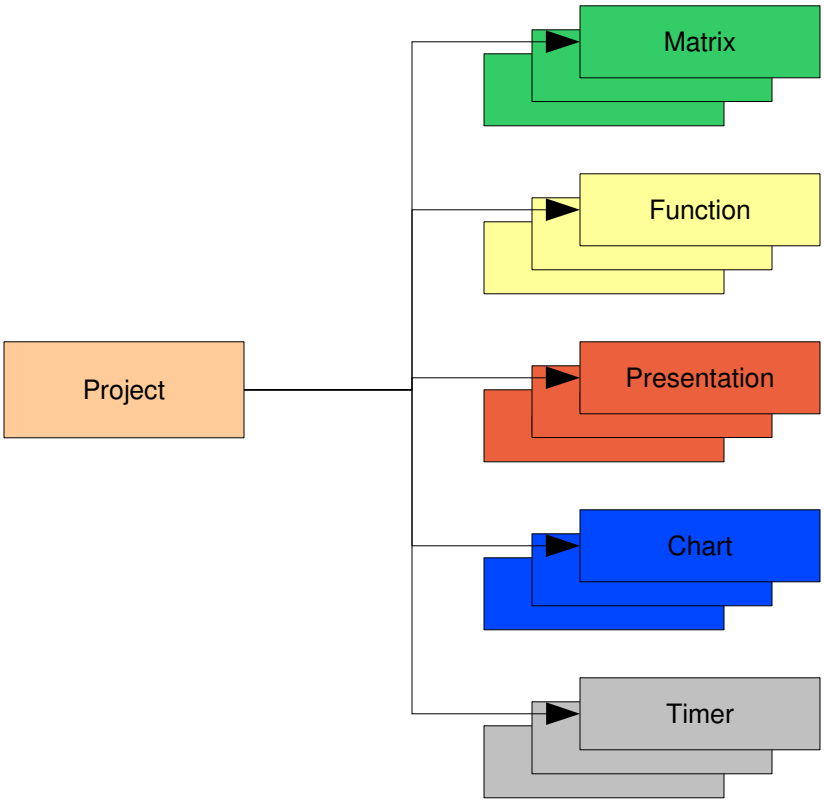
Each machine can run **several projects**:



A project contains all the items to solve a specific problem. In detail, a project contains:

- **matrices** (Matrix class+subclasses)
- **functions** (FunctionHandler)
- **presentations** (Presentations class)

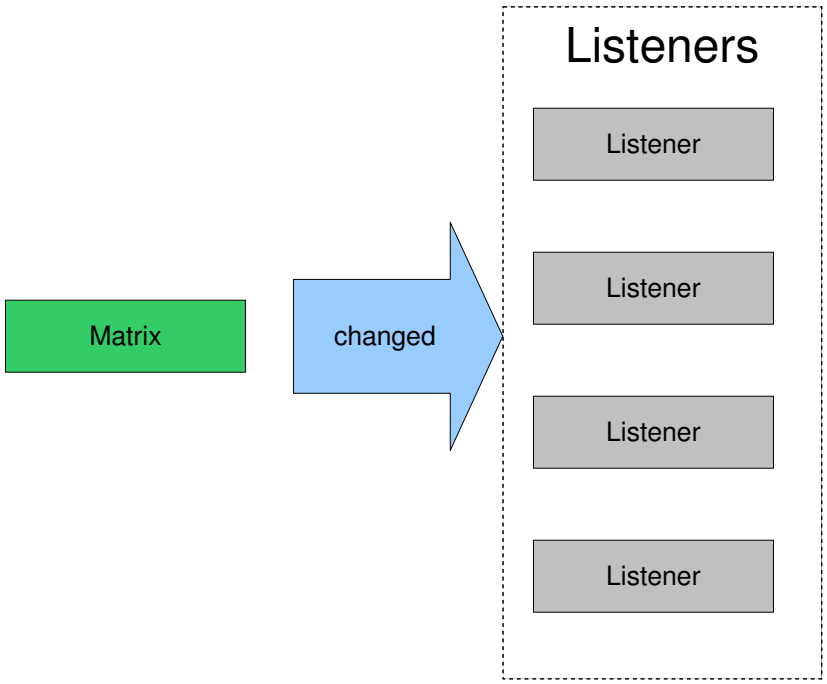
- **charts** (Chart class+subclasses)
- **timers** (Timer class):



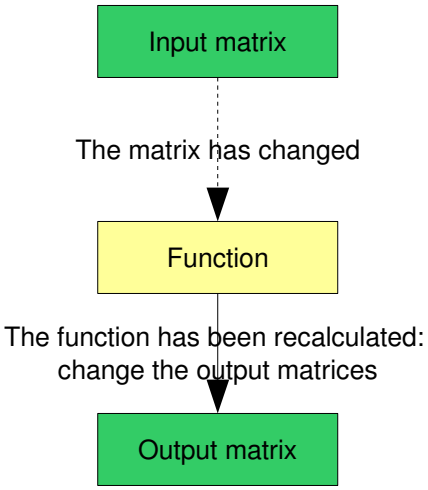
The API module items connections

The items in a project are connected with each other:

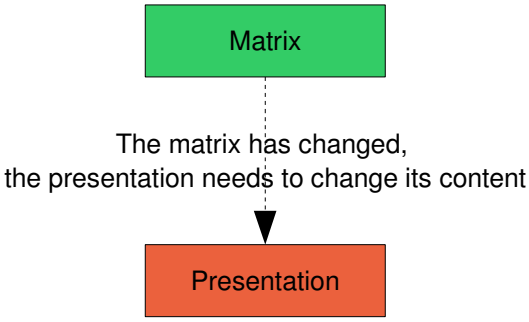
- **directly** one objects contains references to other objects and knows them.
- through the **observer/notifier** pattern (same as the event listeners in java). For example, there are items in the project (functions, charts, presentations) which need to be recalculated or refreshed when a matrix changes its content. To do that, they **register** themselves as listeners of the matrix; when the matrix changes the matrix notifies them with an **event**.



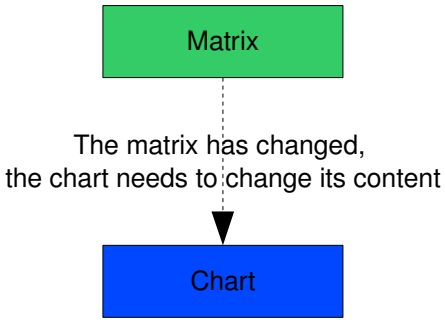
In the following picture we see how a **function** is connected to its input and output matrices. The function originally register as listener to each of its input matrices. When one of the input matrices changes its content, it sends an event to the function. When this happens the function recalculates and changes directly the content of its output matrices (no observer/notifier event here: the function knows its output matrices).



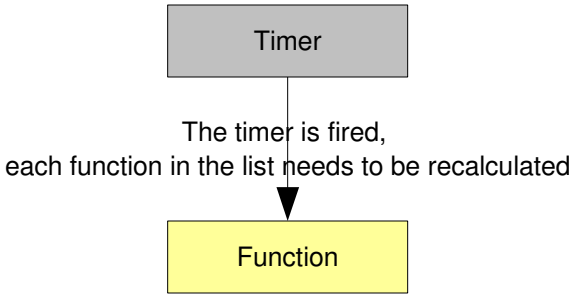
If a **presentation** is registered as listener of a matrix, because the matrix is included in the presentation, when the matrix changes its content the presentation is notified and can be updated.



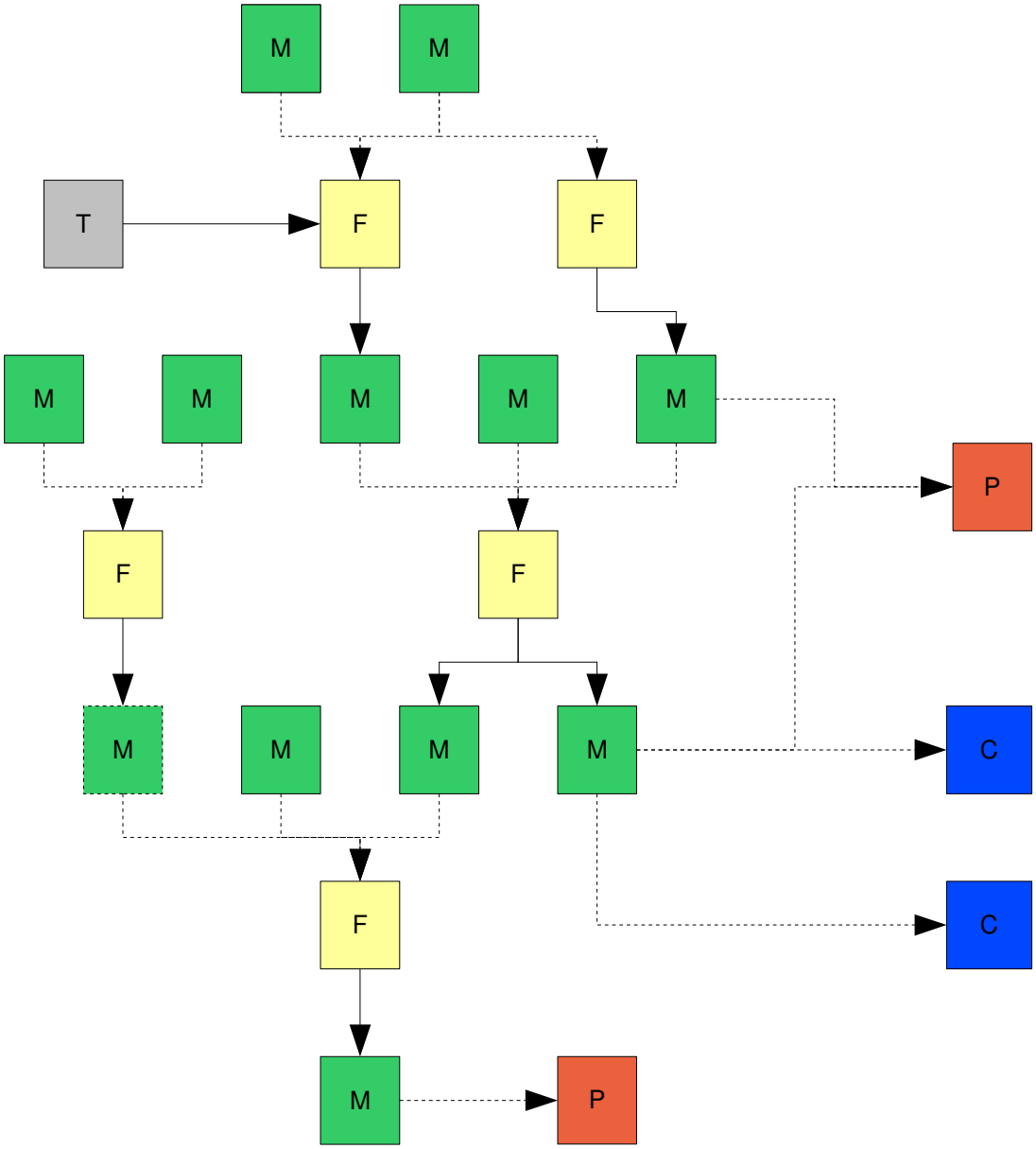
If a **chart** is registered as listener of a matrix, because the matrix contains the data for a series of the chart or for the labels of one of its axes, when the matrix changes its content the chart is notified and can be updated.



When a **timer** fires it triggers the recalculation of all the functions that are in its list.



As a summary, it follows an example of the typical internal structure of a Matrex project:



The structure is a **network of functions and matrices**; the other items are on the limits of the network because:

- presentations and charts only receive the matrices notifications
- timers only fire functions.

The functions calculation order

The **order** in which the functions are calculated can be difficult to establish, expecially for big projects. A Matrex project can be configured to work with or without threads.

- If the project works with threads, it means that each **function calculation is run as a separate thread** of a thread pool. In this case the order in which the

functions are calculated can vary and depends only by which matrices change after each calculation.

- In a no-thread project, the order in which the functions are calculated is also determined by the order in which the functions have registered themselves as listeners of the matrices.

The API-GUI modules items connections

As we told, the API module is **not dependent** by the GUI module, it can potentially work without GUI (for example with a command line program).

To keep the API module independent, the **observer/notifier** pattern is used also between API and GUI: the GUI module acts directly on the API module (add, remove items, save their content) and its editors, views and trees are notified when some items are changed:

- **Project GUIs** (in the main window of the desktop application) update projects adding, updating and deleting items; they are notified when items are added, updated or deleted. In this way the project GUIs are correctly updated if the project is shared.
- **Machine GUIs** (in the main window of the desktop application) update matrices adding, updating and deleting templates; they are notified when templates are added, updated or deleted. In this way the machine GUIs are corrected updated if the machine is shared.

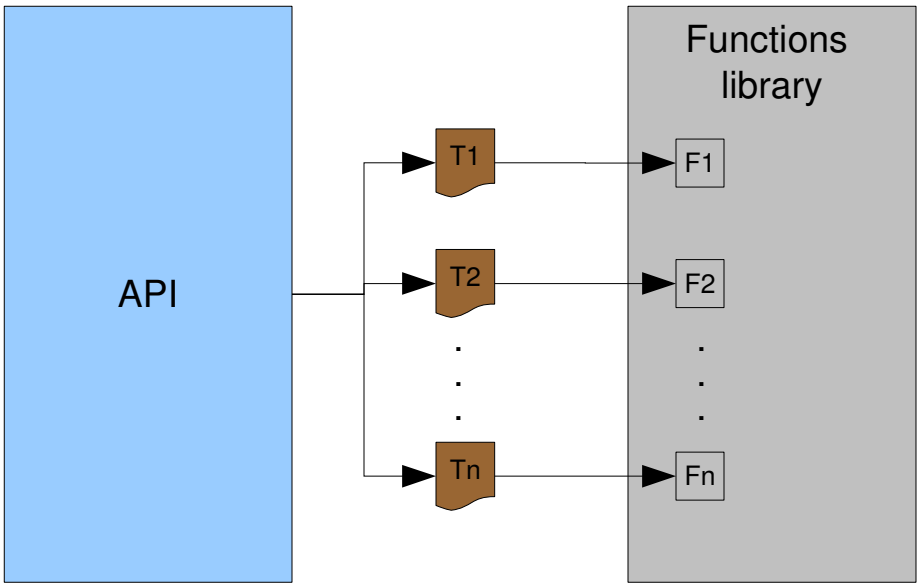
- **Editors** for matrices, functions, presentations, charts and timers save (or anyway update) the content of the related items in the API.
- **Viewers** for matrices, presentation and charts register as listeners to the related items when they open and are notified when the items are saved or updated, to show the changes.

How the API module uses the Functions library

The API module does not have any reference to the Functions library. That gives the possibility, for example, to use a library with different functions together with or *instead* of the given one.

The API module is able to load the Functions library classes and to call them because the **templates descriptions files** contain references to the classes (name+package).

So, since the Matrex engine reads these template description files when it starts, the API knows which classes to call when a function needs to be calculated.



T = template, F = function code