MAF3 Architecture: a brief description

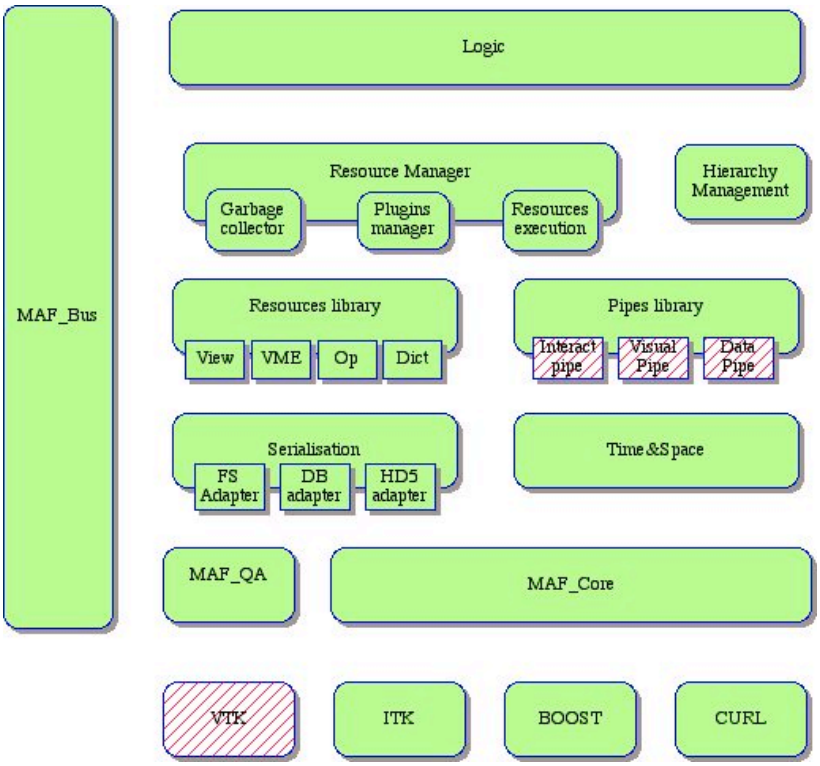
Marco Viceconti and Paolo Quadrani

BioComputing Competence Centre, Bologna – Italy

Rationale

This document provide an brief overview of the MAF3 modular architecture, defining the primary modules that we plan will compose it, and providing a brief description for each of them. To date (August 2009) MAF3 is at the stage where the general architectural design is completed, and the development is starting. Of course the final architecture might be influenced by the development process, also given that some design assumptions are based on methods we never tested before; however, at the current stage this is the best possible description of what MAF3 will be.

Overview



MAF is an application framework for the rapid development of computer aided medicine. The new version of MAF, named MAF3 will retain the general philosophy of the framework, but will take radical departures from the current implementation. MAF3 will still be developed in C++, due to performance constraints; however, we shall move from a mono-threaded monolithic code to a collection of independent modules running on separate threads. In addition, MAF3 will be wrapped in many languages; while we plan to develop a C++ logic module (see following for details), in principle nothing prevents others to develop alternative logic modules using Java, phyton, C#, etc. the idea is to transform MAF into a set of faceless modules that should be relatively easy to compile on any platform, atop of which one can use very different programming environments to define the application logic and the user interface.

A MAF3 application will be a collection of *resources* orchestrated by a pre-defined but extensible *application logic*. The resource manager will dynamically manage resources, available as permanent entities stored in the relative library modules, or as objects pluggable at run-time. Additional modules will provide specialised services to manage resource hierarchies, their serialisation, and their time&space functions. The framework is completed by core modules for quality assurance and code classes definition, including some

third-party libraries that provide foundational functionalities. All resources and modules communicate with each other via a common framework bus, which transports the messages. Each module subscribes to certain types of messages, according to the observer pattern.

There are four types of resources in MAF: data resources, called Virtual Medical Entities or *VME* for short; *View* resources, which provide high level interactive visualisation components; *Operation* resources, which create new VME or modify existing ones. A fourth type of resource, new in MAF3, is the *dictionary* resource: this defines dictionaries and ontologies that can be used to annotate any MAF resource, both for internal use, or for external consumption.

Resources are high-level, user-oriented objects. In order to simplify the re-use of algorithmic components, another type of object can be plugged in MAF3: the *pipe*. The MAF pipe is strongly inspired by the VTK pipe. It is a data flow object, which transform VME into VME (data pipes), VME into render objects (visual pipes) or devices into interactions (interaction pipes). Every module indicated by a separate box in the diagram above will be implemented with a *facade* class that will hide all internal methods, providing a public API as only mean to access its services. Modules will be made available as dynamically loadable libraries, wrapped in as many languages as possible, so to make their use and re-use very easy. The idea is that every programmer will be able to use the whole MAF or only some parts of it in the development of his/her computer aided medicine application. To do so he/she will need only to learn a small set of well documented API instructions, accessible from the language of choice. Each module will execute in a separate thread, and run-time modules will be pluggable only through a programming-by-contract interface. These two features will ensure that plug-ins are accepted only when compliant, and even if one crashes, the rest of the application should be able to continue its execution.

The MAF3 architecture is looking forward: the separation of the serialisation will make possible to easily extend the framework to any biomedical data cloud model; the use of multi-threading and of messaging bus will enable complex deployment where various modules are executed on different CPU, or across a computing cloud.

Foundation Libraries

These are libraries that provide MAF with fundamental functions that would be otherwise too laborious to provide natively. Since the API of the foundation libraries can be called virtually everywhere inside the MAF3 code, these libraries should be developed enforcing a strong back-compatibility policy over their API. Still, the inclusion of these libraries should be done on a frozen-version basis, and only major versions of MAF3 might involve also other versions of the foundation libraries.

A special case will be done for VTK. While VTK is a fundamental component of MAF, its developers decided long ago to avoid enforcing backward compatibility; as an effect many new versions of VTK in the past have been strongly disruptive. Thus, only two modules will contain VTK code: data pipes and visual pipes, and we shall create adapters that convert data objects and render objects on the fly from MAF to VTK. The same strategy might be adopted for other foundation libraries. In any case, the QA framework will monitor the level of encapsulation for each module by counting the number of invocations to foundation libraries in the source code.

MAF QA

MAF_QA includes all utilities for the software Quality Assurance during development of MAF. Examples of functions we plan to include in this module are CPU, memory and I/O profiling functions, modules interdependency tracking, contracts to be used in contract programming, logging and debugging functions, etc.

MAF Core

This module define the core objects needed by all the other modules to allow parameters exchange and the redefinition of base types useful to hide foundational switches, i.e. between unicode and non-unicode.

Serialization

This module is used to abstract the serialisation infrastructure; this will make very easy to switch from file system to database to procedural (i.e. DICOM) serialisation, and possibly to more complex types of storage cloud. The serialisation module operates atomically on each single resource. If there is a need to serialise composite resource, such as a VmeTree, it is the caller class that is responsible of traversing the Tree, and request a serialisation for each VME and for the structure itself.

It will be possible to extend the types of serialisation target by adding adapter sub-modules, probably designed as plug-ins. The module will support both synchronous and asynchronous serialisation, making possible in a general way mechanisms that in MAF2 were hard-coded, such as pyramidal I/O for out-of-core datasets.

Time & Space

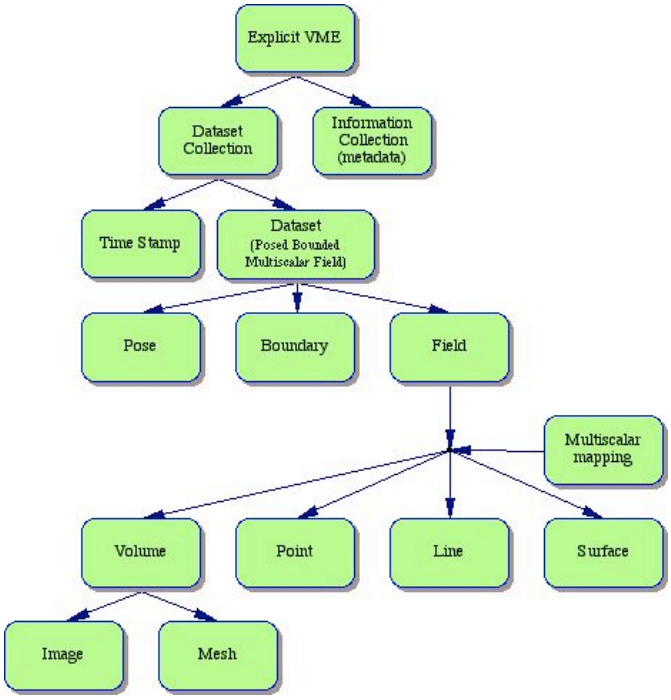
This module generate discrete bases for space and time, such as timers, space-time vector bases and relative interpolators, affine spatial transformations, synchronisation operations, time-stamping, linear algebra operations on pose and time matrices. The idea is to concentrate all space-time discrete operations here, so to make these two concepts highly abstracted in the rest of the framework.

Resources and pipes

Resources and pipes can be added to the framework at run-time using the plug-in mechanism. However, the standard distribution of the framework will include a core of resources and pipes, organised into two separated modules, that provide most essential visualisation and processing tools, and a concrete implementation of the MAF3 Data Object Model (DOM).

Data Object Model

The MAF3 data objects model is represented in the diagram below. The detailed discussion on this DOM goes beyond the scope of this document. Here, it is sufficient to say that with it virtually any biomedical dataset that is defined over space/time will be manageable. The separation of the time stamps, of the pose, the boundary and the actual field already proved in MAF2 essential for efficiency and generality in the manipulation of heterogeneous collections of biomedical data of large dimensions.



Resource manager

At this stage it is not clear if the resource manager will be a single module or the collection of the three indicated sub-modules. In both cases, it will take care of managing the application resources as they are created, destroyed, imported, exported, and interact one to each other.

Hierarchy management

One of the key features of MAF2 is the ability to manage large collections of VME into a hierarchical tree, called VME Tree. In MAF3 we would like to generalise this idea: the hierarchy management module will create and manage hierarchies of resources, of whatsoever type. This will make straightforward the creation of user interface elements to manage large collections of data, views and operations.

Logic

Over the years MAF has defined a generic application logic that prescribes how the user interacts with the various resources. For example in MAF one first selected the VME, and the choses the operation that he wants to apply to it. All these mechanisms are encoded within the framework; the application develop can simply buy into the MAF logic, or customised in part of in full to the specific needs of his application.

MAF Bus

The communication model between modules will be based on the observer pattern. However, to standardise and simplify the use of this communication layer, in MAF3 we plan to add a dedicated module called the MAF Bus. This module will encapsulate all mechanisms for message transport, deliver, logging, etc. In addition, it will provide a standardised message object, that all modules will simply use, ensuring a centralised management of a vital framework component.