CamiTK : a modular framework integrating visualization, image processing and biomechanical modeling

Céline Fouard and Yannick Keraval and Emmanuel Promayon

1 Introduction

Computer Assisted Medical Intervention (*CAMI*) is a transverse field which requires the collaboration of experts in several fields as various as medicine, computer science, mathematics, instrumentation, signal processing, mechanics, modeling, automatics and optics.

CamiTK proposes a common framework for many fields of CAMI so that scientists from one field can develop their own expertise and use the tools developed by other experts without changing application or data structure. This is achieved through extensibility and modularity of the framework which allows every user to develop their own piece of software and use any other developed pieces thanks to CamiTK abstraction layer.

CamiTK is an open-source, cross-platform system, written in C++, that enables developers to prototype efficiently a Computer Assisted Medical Intervention application. It is a generic tool which displays medical imaging, surgical navigations and biomechanical simulations.

2 Contributions

A common environment for CAMI requires data management, algorithms on data, human-machine interface and scientific data visualization. While data types may change from one field to another, as well as algorithms, Human-Machine Interface (HMI) can be generalized for a common software. As it would be problematic to impose one operating system to research scientists of various fields and area of com-

Céline Fouard, Yannick Keraval, Emmanuel Promayon
UJF-Grenoble 1 / CNRS / TIMC-IMAG UMR 5525, Grenoble, F-38041, France e-mail: Celine.Fouard@imag.fr, Yannick.Keraval@imag.fr, Emmanuel.Promayon@imag.fr

petences, the choice has been made for CamiTK to be multi-platform. This requires to choose basis libraries to also be multi-platform and multi-compilers (the most efficient way to achieve this being to choose only Open Source libraries). For those reasons, we choose to base CamiTK on two multi-platform open source libraries: Qt for HMI and Vtk for scientific data visualization. Technical details and how to get your version of CamiTK are given in Table 1.

CamiTK general principle is inspired by component-based software engineering (CBSE). Instead of building an application by adding features on existing code, CBSE intends to build a software by integrating, arranging or assembling pre-built software components. CamiTK shares the CBSE guiding principles : *i)* reuse and do not reinvent the wheel and *ii)* assemble pre-built modules and components instead of adding codes where needed. This can be very efficient when standard component exists. Unfortunately no such standard exists in CAMI field. CamiTK therefore introduces simple light standards for software modules that can be found in CAMI application. This light standards are not specific to any sub-domain such as imaging, robotics, surgical navigation or simulation. To conform to this standards, a software component simply has to inherit from one of the four CamiTK extensions.

The four possible types of extension are *component* (data management), *viewer* (data visualization), *action* (algorithms), and *application* (the HMI). Figure 1 shows several examples of existing extensions. CamiTK source code includes a list of commonly required extensions: volume image input/output management through several format and mesh management, a classical medical image viewer with in four zones (axial, coronal, sagittal and 3D view). Among available *action* extensions, we developed volume image filtering algorithms using ITK, model deformation using Sofa, ArtiSynth or ANSYS, and 3D mesh reconstruction from stereoscopic video. Some basic *application* extensions are also provided with CamiTK.

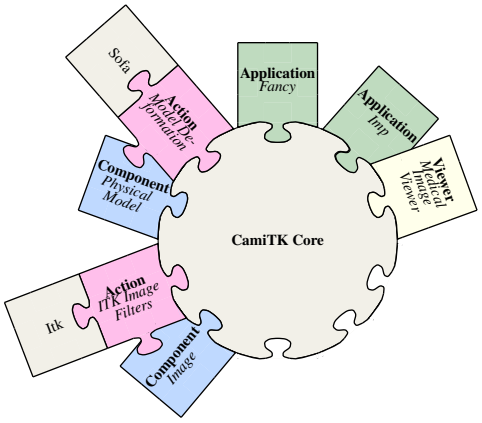

Fig. 1 CamiTK architecture: several existing and collaborating modules.

URL	http://camitk.imag.fr
License	The core is available under the terms of the GPL v2
Language	C++
Size	≈ 200 classes, 35.000 Single Lines Of Code
Documentation	API documentation, user tutorials, developper tutorials, mailing lists
Dependencies	Qt, VTK, optional: libxml2, xerces-c, ITK
Installation	Windows setup, Ubuntu package, source package, svn (through the collaborative forge)

Table 1 CamiTK Factsheet.

3 CamiTK Architecture

CamiTK core architecture follows the Model-View-Presenter design pattern, a generalization of the Model-View-Controller design pattern where the presenter acts more like a supervising controller.

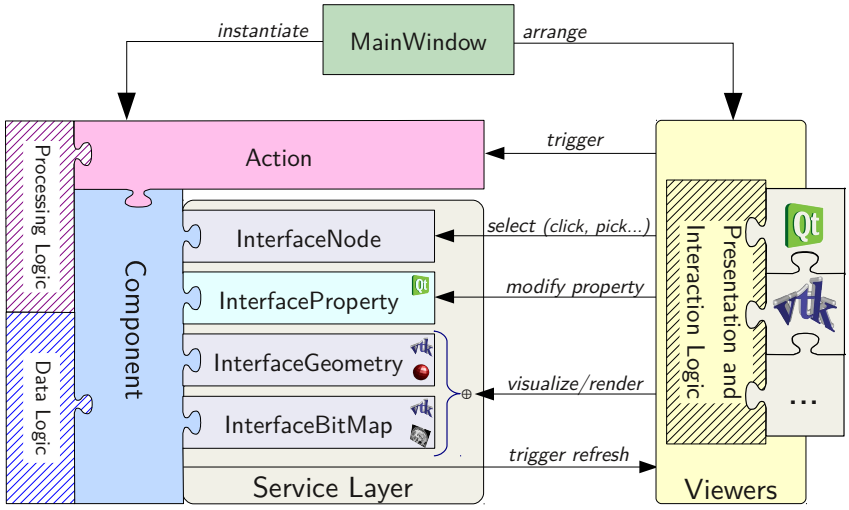

Fig. 2 CamiTK Architecture Overview.

In classical software architecture, MVP design pattern helps to divide the logic in two: domain logic and application logic. Domain logic, the Model, is the part of the data structures and algorithms that is directly linked with the research field or sub-domain. Application logic is the part generally used to build a Graphical User Interface (GUI) to present and interact with the domain logic data and algorithms. Application logic generally aggregates the View and Presenter parts of MVP. CamiTK lets the researcher concentrates on the domain logic and ease the development of the application logic by following CBSE principles. To do so, a software

based on CamiTK assembles what we called *viewers* (application logic) than are used to present and interact with *components*, i.e. the data (data logic), and *actions*, i.e. the algorithms (processing logic). *Viewers* and *components* are binded together by a *main window* to define an interactive application. Figure 2 presents an overview of the CamiTK architecture.

CamiTK furthermore distinguishes the domain logic in data logic, i.e. the management of dynamic or static data structures, and the processing logic, i.e. the modification and transformation of data. This is an essential distinction as it facilitates software testing and improves the segmentation of tasks. The data logic is handled by what we called *components*, while the processing logic is handled by what we called *actions*. In order to easily glue together viewers, components and actions, CamiTK uses a Service Layer Pattern. The service layer is classically linked to the viewers by an Observer Pattern.

The service layer simplifies the creation of new components and actions by generalizing concepts and predefining generic behaviors. The service layer is an overlay of the `Component` class (see Figure 2). The `Component` class obeys to four interfaces: `InterfaceNode`, `InterfaceProperty`, `InterfaceGeometry` and `InterfaceBitMap`. Each interface describes the capabilities of a component without committing it to use a specific implementation. It can also be thought as a contract where the component is the provider of services. On the other side of the MVP, the viewers are consumers of the services. The viewers interacts with the components without knowing exactly which kind of data is managed by the component or how it will react to a specific call.

Adding an extension mainly consists in writing a new C++ class derived from one of the CamiTK core extension classes (which can generally be done in 10-50 Single Lines Of Code). Core extension classes already define a lot of default behaviors, each of them can be redefined by just overriding the corresponding method(s).

CamiTK also follows some rules of test-driven development by providing a default unit test architecture for its components. An automated quality software process is currently being set up for CamiTK thanks to Kitware CDash.

4 Conclusion

Our framework could be seen as a linking module that can fill the gap between highly specialized software covering different CAMI research fields.

The first public version of CamiTK was published in october 2011 (version 2.0), anybody wanting to contribute to join the effort is welcome. The current work is focusing on simplifying the scripting of actions and on software quality (test, packaging, continuous integration).

We are interested in participating to any workgroup in order to increase interoperability between CAMI software.