The MITK Approach

Ivo Wolf, Marco Nolden, Thomas Böttger, Ingmar Wegner, Max Schöbinger, Mark Hastenteufel, Tobias Heimann, Hans-Peter Meinzer, and Marcus Vetter

Div. Medical and Biological Informatics/B010,
German Cancer Research Center, Heidelberg, Germany
email: i.wolf@dkfz.de*

Abstract. The Medical Imaging Interaction Toolkit (MITK) is an open-source toolkit for the development of interactive medical image analysis software. MITK is based on the open-source Insight Toolkit (ITK) and Visualization Toolkit (VTK) and extends them with features required for interactive systems. ITK is used for the algorithmic scope and general infrastructure, VTK for visualization. Key features of MITK are the coordination of multiple 2D and 3D visualizations of arbitrary data, a general interaction concept including undo/redo, and its extendibility and flexibility to create tailored applications due to its toolkit character and different layers of hidden complexity. The paper gives a brief introduction into the overall concepts and goals of the MITK approach. Suggestions and participation are welcome. MITK is available at www.mitk.org.

1 Introduction

Today, the development of interactive software for medical image analysis is supported by either extendable applications (e.g., 3D-Slicer, Analyze, Amira, VolView) or complete development environments (e.g., AVS, Khoros, MeVisLab, SCIRun). In contrast, the MITK approach is to provide a *toolkit* on which applications – for specific problems or general use – and development environments can be built upon.

ITK [1] and VTK [2] are successful and well-designed toolkits for the algorithmic and visualization aspect of medical image analysis. Coordination of different types of visualizations and interactions, both important for imaged-based medical software, are not within their scope. MITK adds support for this and other features common to interactive systems. The goal is that specific realizations of the added aspects (e.g., interactions) can as powerfully be combined as this is the case for algorithms and visualizations in ITK and VTK (section 2). Aspects covered by ITK or VTK are re-used from these toolkits (section 3). Different layers of hidden complexity allow re-use in-the-small and in-the-large to support the development of applications with highly specialized features as well as mainstream tasks (section 4).

* Partly supported by the Deutsche Forschungsgemeinschaft (DFG) within the project SFB 414 "Information Technology in Medicine – Computer and Sensor Supported Surgery".

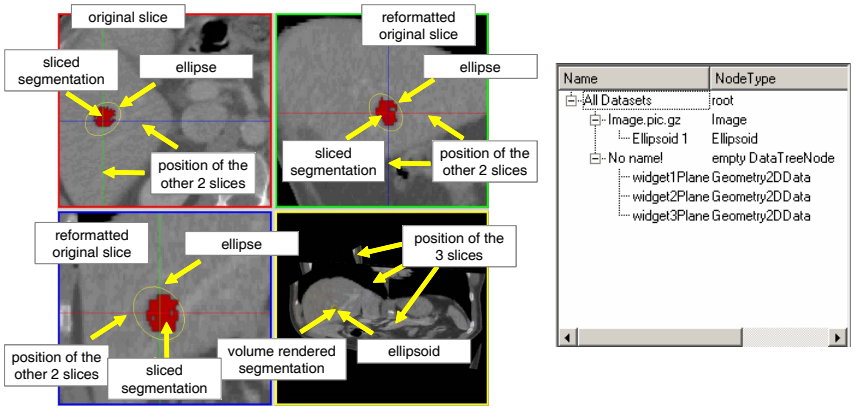

Fig. 1. Left: Four consistent views of 6 data objects (original image, segmentation, ellipsoid, 3 slices) require coordination of 20 scene-graph objects in four scene-graphs. In MITK, only the data objects need to be added to the MITK data tree (right).

2 Coordination of visualizations and interactions

Most object-oriented visualization libraries use a scene-graph concept to define the contents of a display area. Consequently, different contents require different scene-graphs. From this point of view, the four display areas of the application shown in figure 1 have four different contents requiring four different scene-graphs and altogether (at least) 20 objects within the scene-graphs. On the other hand, the four display areas all show the same data, but in different ways (orthogonal reformations, 3D view).

MITK uses a so-called *data tree* to define the contents of multiple (or all) display areas, instead of an individual scene-graph per display area. Thus, the central elements, on which visualization and interaction objects act, are the *data*, leading to a *model-view-controller*-like design of MITK. Changing a data object, or its position, orientation, color etc. results in a consistent change in all display areas using the respective data tree.

Interactions of the user with the data are performed directly on the data. This ensures a consistent behavior in all views and enables to define the interaction (in many cases) regardless whether it is done in a 2D or 3D view or with a 3D input device. Since more complex interactions with data consist of several states, interactor-classes are realized as *state-machines* in MITK.

MITK takes care of the conversion from input coordinates (e.g., position of the mouse cursor) to 3D world-coordinates in a coordinate-system in millimeters (not pixels!). Conversions to/from intrinsic coordinates of a data object (e.g., pixels in case of an image) are uniformly performed using so-called *geometry frame* objects [3], which are used to position data objects in space and define the area or volume to be rendered (e.g., reformatted slice, including curved reformations [4]).

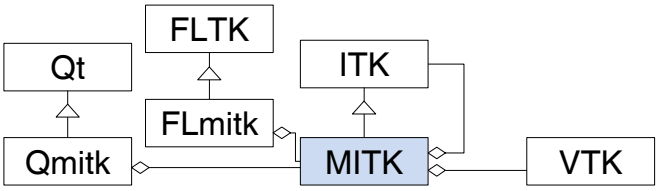

Fig. 2. Illustration of MITK's relation to ITK, VTK and GUI toolkits. Most MITK classes are derived from ITK, whereas VTK classes are only used. MITK itself is independent of the GUI library. GUI dependent parts contain optional add-ons, which are currently implemented for Qt (Qmitk). A small example for FLTK exists.

3 Re-use of open-source code and concepts

One advantage of open-source software are the excellent possibilities of re-use and extension. Our general idea is to write algorithms for ITK, visualization methods for VTK, and organize both including user interaction in MITK. Thus, MITK re-uses virtually everything available from ITK and VTK. In contrast to most other software that simply uses ITK and/or VTK, MITK's way of re-use is much deeper.

Most MITK classes are derived from ITK classes, which allows to re-use the smart-pointer-, time-stamp-, observer-, and debugging-mechanisms of ITK. MITK's data objects contain either ITK or VTK data objects and methods are provided to access the data from the respective other toolkit. To allow different visualizations of data objects, e.g., 2D and 3D views, MITK uses a variant of VTK's idea to have a so-called *mapper* object for creating graphic primitives for each data object. Internally, VTK is used, coordinated by MITK to keep the displays consistent.

Pipeline concepts have proved to be a powerful way to combine algorithms, especially for interactive applications. Instead of inventing yet another pipeline concept, the pipeline of ITK is used in MITK, simply by deriving MITK's data and algorithmic classes from the respective ITK base-classes. Most algorithmic classes in MITK are encapsulations of ITK or VTK algorithms for specific tasks, to adapt them to MITK's unified coordinate system, and/or to support time-resolved data.

Furthermore, the software process and most principles of the ITK/VTK style guide are adopted for MITK. The relations of MITK to ITK and VTK are illustrated in figure 2. Like ITK and VTK, MITK is an object-oriented, cross-platform library implemented in C++.

4 Layers of hidden complexity

By definition toolkits aim at providing kits of tools to assist building individual applications. Pre-defined components for recurring tasks can accelerate development, but may reduce the capability to create specific applications. The goal

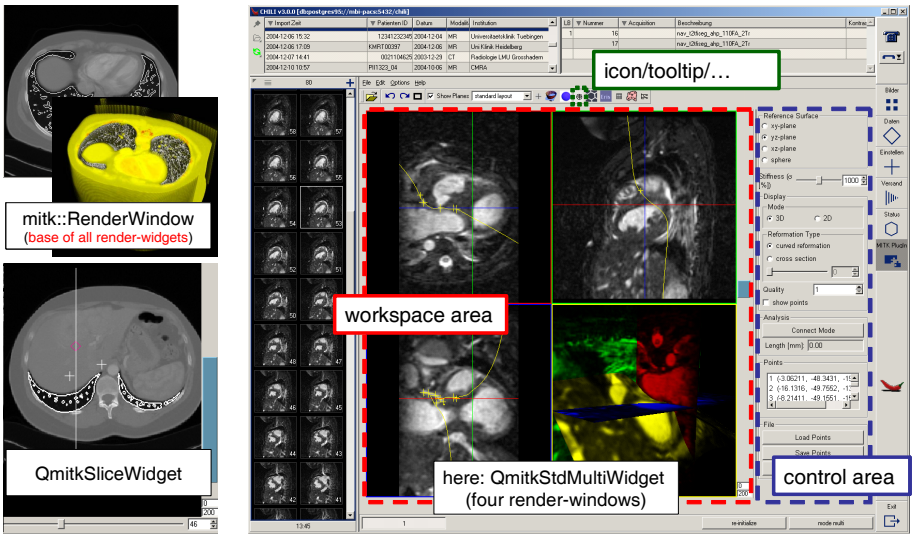

Fig. 3. Widgets on different layers of hidden complexity within MITK. All visualization takes place in render-windows (top left), higher-level widgets for visualization just add control widgets (bottom left) and/or contain several render-windows (central part of right). Right: MITK *functionalities* used within an MITK-based plug-in for the PACS/telemedicine system *Chili*.

within MITK is to provide different layers of pre-defined components that hide more and more complexity. Small-scale components can flexibly be combined, larger entities solve common tasks, inevitably at the expense of flexibility.

The basic MITK component for display is called *render-window* (as in VTK). All kinds of visualizations can be rendered into a render-window (figure 3, top left), thus all higher-level widgets for visualization contain (one or more) render-windows and just add widgets or convenience methods for controlling the visualization. One example is the `QmitkSliceWidget` for 2D-slicing through the data, see bottom left of figure 3. The slicing is controlled by a controller class, which is invisible and GUI-independent (`mitk::SliceNavigationController`) [3]. The currently selected slice number is being displayed using a widget connected to the controller class, here a slider and a spin-box (`QmitkSliderNavigator`, GUI-dependent). Another example is a combination of four render-windows with various layouts (`QmitkStdMultiWidget`, central part of figure 3, right).

So-called *functionalities* are currently the top-most layer of hidden complexity within MITK. Functionalities facilitate the structured combination of modules. A functionality is a module for a specific task, combining a user interface with algorithmic function. It consists of (figure 3, right)

- an identification (name of the functionality, icon, tooltip, ...),
- a workspace area, where the main interaction is taking place,

– a control area containing GUI elements to set parameters, and
– the algorithmic implementation.

Communication between functionalities is largely based on the data tree. Each functionality accesses the data objects contained in the tree, changes them and/or creates new data objects. Other functionalities can continue to work on the changed and/or newly created data tree entries. By that, functionalities can communicate without becoming dependent on each other.

Furthermore, an add-on to MITK allows to create plug-ins for the PACS/telemedicine system *Chili* (Chili GmbH, Heidelberg, Germany, http://www.chili-radiology.com). This facilitates the clinical integration of the software, since the radiologist or surgeon using the MITK-based software find it seamlessly integrated into the workspace he/she is used to.

5 Conclusion

MITK aims at supporting the development of interactive systems from the *toolkit level*. MITK is *not* intended as an application framework, but there are some *optional* application-level add-ons to MITK (e.g., functionalities, plug-in integration). MITK allows the construction of applications specifically tailored for a medical task, providing only those features to the user (physician) that are required. Another advantage is that the toolkit can be used within existing software.

MITK is available from www.mitk.org via cvs-access or as a zipped-archive (current version is 0.2). Although already in use for a variety of applications, there are still lots of ideas for improvements. We soon will add more high-level modules, e.g., for common tasks as volume-rendering, cropping, or creation of isosurfaces. Additional layers of hidden complexity may be useful, e.g., visual programming, or usage of visually created pipelines exported from other tools. We are open for suggestions. Participation is welcome.

References

1. Ibanez, L., Schroeder, W., Ng, L., Cates, J.: The ITK Software Guide. Insight Software Consortium (2003)
2. Schroeder, W.J., Martin, K.M., Avila, L.S., Law, C.C.: The Visualization Toolkit User's Guide. Kitware, Inc. (2003)
3. Wolf, I., Vetter, M., Wegner, I., Nolden, M., Böttger, T., Hastenteufel, M., Schöbinger, M., Kunert, T., Meinzer, H.P.: The Medical Imaging Interaction Toolkit (MITK): a toolkit facilitating the creation of interactive software by extending VTK and ITK. In Galloway, R.L., ed.: SPIE Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display. Volume 5367. (2004) 16–27
4. Wolf, I., Hastenteufel, M., Wegner, I., Vetter, M., Greil, G., Küttner, A., Claussen, C.D., Meinzer, H.P.: Curved reformations using the Medical Imaging Interaction Toolkit (MITK). In Galloway, R.L., Cleary, K.R., eds.: SPIE Medical Imaging 2005: Visualization, Image-Guided Procedures, and Display. Volume 5744. (2005) 831–838