IGSTK: A State Machine Architecture for an Open Source Software Toolkit for Image-Guided Surgery Applications

Luis Ibanez[1], Julien Jomier[2], David Gobbi[3], Rick Avila[1], M. Brian Blake[4], Hee-su Kim[6], Kevin Gary[5], Stephen Aylward[2], and Kevin Cleary[6]

[1] Kitware Inc., Clifton Park, NY, 12065, USA
luis.ibanez@kitware.com, rick.avila@kitware.com
http://www.kitware.com/
[2] Computer-Aided Diagnosis and Display Laboratory, University of North Carolina, Chapel Hill, NC, 27599, USA
jjomier@cs.unc.edu, aylward@unc.edu
[3] Atamai Inc., London, Ontario, N6B 2R4, Canada
dgobbi@atamai.com
[4] Department of Computer Science, Georgetown University, Washington, DC, 20007, USA
blakeb@cs.georgetown.edu
[5] Division of Computing Studies, Arizona State University, Mesa, Arizona 85212, USA
kgary@asu.edu
[6] Imaging Science and Information Systems (ISIS) Center, Department of Radiology, Georgetown University Medical Center, Washington, DC, 20007, USA
cleary@georgetown.edu, hkim@isis.imac.georgetown.edu

Abstract. The Image-Guided Surgery Toolkit (IGSTK) is an Open Source software project being developed under NIH/NIBIB funding. The toolkit will provide a common platform for implementing image-guided surgery applications and for fostering research in the field. The toolkit is based on several other open source toolkits including ITK, VTK and FLTK. Its architecture is based on the use of medium size classes, each one of them containing an explicit State Machine and a minimized API that enforces fault tolerance by design. This paper describes the architecture and its rationale.

1 Introduction

The image-guided surgery toolkit (IGSTK) is an open source project aimed at providing robust software for developing image-guided surgery applications. Image-guided surgery (IGS) involves the use of pre-operative medical images to guide instruments during minimally invasive procedures.

IGS systems have been commercially available for about ten years, but the development of such systems is still an active area of research. This research includes image analysis method development as well as the exploration of new clinical applications.

Image-guided surgery systems, however, require significant programming time and expertise to prototype and deploy. In addition, an IGS system must be well-tested since it will be used in a patient-critical environment. These programming and validation requirements challenge all forms of IGS systems research.

The IGSTK project addresses the challenges of IGS systems research. The toolkit contains the basic software components to construct an image-guided system including a tracker and a default graphical user interface that includes a four-quadrant view with image overlay. The highest priority in the design of the toolkit has been robustness and quality. The toolkit is based on the existing open source components, such as ITK, VTK, and FLTK [8,9,10]. One of the early IGSTK architecture decisions was to use State Machines for all of the key software components. The state machines inherently provide fault tolerance and thereby address patient safety.

In this paper, we describe the overall IGSTK software architecture, its state machine, and other key components. Key components that are available in the current beta release include the base state machine, spatial objects, tracker objects, and tracker communications, viewers and logging. Each component is presented, and the paper concludes with a short example application.

The IGSTK toolkit is Open Source software distributed under a BSD-like license equivalent to VTK and ITK licenses [11]. The software is developed following the practices of Agile Programming and therefore is made available from the very early stages of development. Basic information related to IGSTK can be found at the web site www.igstk.org and the Wiki pages http://public.kitware.com/IGSTKWIKI/.

Instructions for configuring and building the toolkit are available at http://public.kitware.com/IGSTKWIKI/index.php/How_to_build_IGSTK. You are welcome to try the software, review the source code and send your comments to the IGSTK development team. The toolkit is currently in Beta form and evolving, and a stable release is anticipated toward the end of the year. Note that this software should only be use in clinical cases under IRB approval. You are allowed to use IGSTK for free in academic and commercial applications but it is your responsibility to perform the tests and validations required by regulatory bodies such as the U.S. Food and Drug Administration (FDA) [4,5,6,7].

2 Architecture

Contrary to what is commonly found in software toolkits, IGSTK professes a very Spartan approach to features and functionalities. Given that IGSTK is intended to be used for developing applications that will run in the surgical room; the nature of such critical applications imposes very particular restrictions on what is desirable or not in the toolkit. IGSTK has been designed in such a way that risk of harm to the patient resulting from misuse of the classes, whether it is by negligence or by accident, is minimized. In order to achieve this goal of safety-by-design the architecture was based on the following principles:

1. Requirements are generated by studying the types of surgical applications where the toolkit may be used in a clinical context.

2. Every component has a minimal set of features. Only features that are necessary for providing functionalities requested in the requirements are implemented.
3. Every component is based on a State Machine abstraction. In this way the state of the class is always known, and precautions are taken for making sure that the class is never set in an invalid state, due to oversight or negligence.

The combination of a restrictive API and the use of State Machines makes possible to implement a high level of software testing and to enforce high quality standards for code coverage and run-time validation. A restricted API is achieved by creating middle size components that put together all the elements needed for providing a specific functionality. In this way, a minimum number of parameters are left free for the application developers to define, and therefore the opportunities for error are drastically reduced [3].

Figure 1 presents a UML collaboration diagram of the major IGSTK components involved in a typical image-guided surgery application.

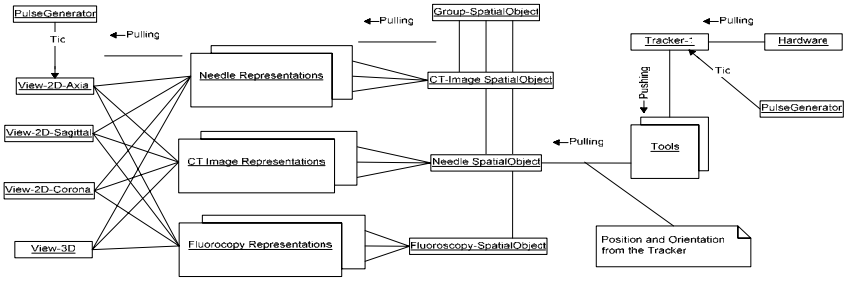

Figure 1 : Architecture of Typical Image-Guided Surgery Application

The elements on the left are those that will be visible to the clinician performing the surgical intervention. Those are the View classes representing the abstraction of a visualization window in the screen of the application. The view classes are a combination of FLTK and VTK classes that restrict user interaction to a set of safe and well defined operations. Each viewer is refreshed at a rate that can be specified by the application developer. The pulses for triggering the refresh of the viewers are provided by a PulseGenerator class indicated in the upper left corner of the Figure. View classes display renderings of the surgical scene that are composed of the minimal number of elements required to provide useful information to the clinician. The actual physical objects are modeled using SpatialObjects, represented in the central part of the Figure. They include images of various modalities as well as simple geometrical shapes such as Cylinders and Ellipsoids. The aspect that is used for

presenting the SpatialObjects to the surgeon is defined by SpatialObject-Representation classes. These classes provide the connection between the Views and the SpatialObject. A representation class defines properties such as color, transparency and the actual rendering methodology used for presenting the object in the scene. Some of the objects in the scene are static, while some of them are moving in space. In the surgical environment, it is critical to track the spatial positions and orientations of some of the surgical instruments. IGSTK provides support for some of the trackers that are commonly used in medical applications. This includes optical and electromagnetic trackers. The role of the tracker class is illustrated in the right side of the Figure. The Tracker updates the position and orientation of a particular spatial object; this object may be representing a surgical needle for example.

3 State Machines

State Machines are a fundamental concept in computer programming. They were introduced by Alan Turing in 1936 [2] as a formalism for supporting his work on determining whether the execution of an algorithm will ever stop or not. This problem is also known as the "*Entscheidungsproblem problem*". A State Machine is defined by a set of states, a set of inputs and a set of transitions from one state to another. A Finite State Machine (FSM) is a state machine where the number of states is finite, and a Deterministic State Machine (DSM) is one where a given input presented to a given state will always led to a unique state. In practice all computers are state machines, unfortunately their possible number of states is so large that they can barely be considered to be FSMs. An alternative way of looking at this large number of states is to assume that some of those states are not modeled and then become random elements on the behavior of the state machine. In this interpretation, the state machine is a Non-Deterministic State Machine. Transitions in a State Machine result in actions being taken. Some State Machine paradigms execute the actions when leaving the old state, while some others do it when entering the new state.

Computer programs, in particular those that are modeled using Object Oriented programming are naturally described in terms of state machines. Unfortunately, the lack of formality in traditional programming leads to under-defined state machines, where the states are poorly defined, and the transitions between states are rarely stated explicitly. Such relaxed programming practices produce programs that behave erratically and unpredictably. Those are exactly the kind of programs that are unacceptable in a critical application such as image-guided surgery. The sake of reliability and robustness in IGSTK lead the development team to select the State Machine model on the very early stages of the project. State Machines are an excellent way of limiting the number of possible behaviors and ensuring that a program will always be in a valid condition, and that all possible behavior has been studied in advance by the developer team in order to guarantee repeatability and deterministic behavior.

A generic State Machine class is available in the toolkit and provides the abstraction of the set of states, the set of inputs and the set of transitions. Each IGSTK component instantiate internally its own state machine and at construction time programs the full behavior of the State Machine. This organization makes possible to anticipate how the classes will work when their methods are invoked in *any* order. It is rare to find object oriented classes that will behave correctly or at least without run-time failures when their methods are invoked in random order.

When using State Machines, it is clear very early in the development cycle that API simplicity is a must for supporting robustness and reliability. In the context of surgical guidance, we must consider *flexibility* and *abundance of features* to be undesirable, because each one of them brings more opportunities for thing to go wrong during the surgical intervention. State Machines make possible to exercise full coverage, not only in the sense of number of lines executed during the testing cycle, but also in the sense of all possible execution paths of the code, at least at a single-class level.

A number of C++-Language features have been used in order to enforce the safety and integrity of the State Machine. For example, the methods that actually perform actions are all declared private and can only be invoked by the State Machine itself. Encapsulation and enforcement of const-correctness are also used at great lengths in order to reduce the risks of misusing the code.

4 Components

The following sections describe some of the main components available in the IGSTK Toolkit. For further details please refer to the online documentation:

http://public.kitware.com/dashboard.php?name=igstk

You will also find useful to look at the design and development discussions available on the IGSTK Wiki:

http://public.kitware.com/IGSTKWIKI/index.php/Main_Page

4.1 Spatial Objects / Viewers

Spatial Objects are the holders of the geometrical representation of physical objects that must be presented to the surgeon. The characteristics of SpatialObject are selected based on their capability for conveying useful information to the clinician. The IGSTK SpatialObjects are classes that encapsulate ITK Spatial Objects inside a restricted API subject to the control of a State Machine. In this way we obtain all the functionality without the risk associated with exposing all the flexibility of the ITK classes. IGSTK Spatial Objects can be attached to a Tracker object, their corresponding State Machines make sure that one Spatial Object is associated one-to-

one to a Tracker, and that once they have been associated, it is no longer possible to change the position of the object programmatically. This again, ensures safety-by-design [3].

Viewers are the classes that take care of presenting the renderings of surgical scenes to the clinician. These are critical since they are in many cases the main source of information that is presented to the clinician, especially in cases of minimally invasive surgery such as endovascular intervention and laparoscopy. Viewers are built using VTK and FLTK classes that combined are encapsulated into a restrictive API and subject to the control of a State Machine. Viewers limit the number of ways in which a user can interact with the scene. These interactions are defined considering that a surgeon may only have access to a touch screen in the surgery room, devoid of a keyboard or a mouse. In other contexts, an assistant may be manipulating the software under the verbal instructions given by the surgeon.

4.2 Tracker / Communications

The IGSTK tracking component was adopted from a set of C++ tracker classes donated to the project by Atamai Inc. [1]. We have reorganized the code to fit our state machine architecture and to connect seamlessly with the spatial object classes described in the previous section.

Incorporation of a finite state machine into the tracker component is perhaps one of the most remarkable aspects of the IGSTK architecture. Since the state machine is a mirror of the actual states of the physical tracking device, it ensures that only commands that the device is ready to correctly respond to will be sent to the device. Furthermore, this safety is provided purely through the design of the state machine transition table, without the need for `if/else` error checking throughout the code. The ultimate goal, as far as code safety is concerned, is for unexpected responses from the device to occur only in the unlikely event of a hardware device malfunction.

The tracking component consists of a Tracker base class, with specialized subclasses to support the support the AURORA and POLARIS tracking devices (Northern Digital Inc., Waterloo, Canada). The Tracker object does not communicate directly with these devices via the serial port, but instead communicates via a SerialCommunication object that acts as a proxy between the Tracker object and the tracking device. The use of this communication proxy serves several purposes:

1. All platform-dependent serial port control code is contained within the communication object, while the Tracker code is fully platform independent.
2. The communication proxy can be replaced with a “simulator” that replays previously recorded serial data streams for testing or demo purposes.
3. The communication proxy can be replaced with an object that sends the data stream to a remote device over a network via TCP/IP.

Our goal for the immediate future is to extend these classes to support multi-threading, so that the tracking information can be passed to the display component at the chosen display refresh rate while, simultaneously, a "safety thread" can run at the full data rate of the tracking system for the purpose of collision avoidance.

4.3 Logging

Logging is a fundamental functionality for software intended to be used in critical applications. In the context of image-guided surgery, logging of the events that occur during an intervention makes possible to retrospectively evaluate the behavior of the system and verify whether the assumptions made during the design and implementation phases of the software still hold in practice or not. For example, if it has been assumed that particular actions of the software would take a small amount of time, this assumption can be verified after a surgery by simply analyzing the logging report.

Logging is also an invaluable help for debugging the software during the development phase. The capability for reporting the state and conditions of particular software components facilitates to identify, isolate and correct for errors that may have been introduced during the development phase, or that may even be rooted in design flaws.

The logging capabilities of IGSTK were considered to be so useful that they were transferred to the Insight Toolkit (ITK). This code transfer is a remarkable example of how a well defined design followed by careful implementation can result in source code that is at the same time efficient, robust and maintainable.

The logging functionalities of IGSTK, now in ITK, are provided by the classes: itkLoggerOutput, itkLogger and itkLoggerManager. A logger acts as a dispatcher that receives messages from IGSTK objects and redirects those messages to LoggerOutputs. The typical examples of Logger outputs are a file and the console. A typical application may contain multiple Loggers. In this way it is possible to have different levels of detail in the reporting and it is also possible to separate the logging of critical activities such as optical and electro-magnetic tracking from the logging of conventional tasks such as timing for reading and writing files. The LoggerManager class allows controlling the behavior of all the Loggers presented in an application.

The classes involved in Logging present a particular design challenge because they are required to be more robust than the rest of the components, if they are to be the ones that report whenever one of the other components fail. They are also required to run very efficiently because they may be invoked from sections of code that are executed hundreds of times per second. The policies for defining the level of detail to be used for logging during a surgical intervention are a matter of open discussion and should be revisited as technology changes, for example, when faster Trackers become available.

5 Example Applications

As set of minimal example applications are available in the source code distribution. These applications progressively introduce the use of IGSTK components starting from a simple combination of a viewer, a spatial object and its representation, and going up to a four quadrant view with geometrical objects and connections to optical and magnetic trackers. Details on these applications can be found in the IGSTK Wiki pages http://public.kitware.com/IGSTKWIKI/index.php/Example_Applications.

5 Conclusions

The IGSTK toolkit demonstrates that high standards of quality control can be achieved on mission-critical applications using Open Source software and modern programming technologies such as Agile Programming. An Open Source approach makes possible to subject the source code to the scrutiny of a larger community of experts and by recovering their feedback it is possible to increase the general quality, robustness and safety of the code.

Open Source repositories make possible to combine the experiences and efforts of multiple research teams into a common resource and from it to foster the advancement of the field since new groups do not have to keep reinventing the wheel.

6 Acknowledgements

This research is supported by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) at the National Institute of Health (NIH) under grant R42EB000374 and by U.S. Army grant W81XWH-04-1-0078. The content of this manuscript does not necessarily reflect the position or policy of the U.S. Government. The authors would like to thank the IGSTK advisory board for their advice throughout the project: Will Schroeder of Kitware; Ivo Wolf of the University of Heidelberg; Peter Kazanzides and Anton DeGuet of Johns Hopkins University; and Ingmar Bitter, Matt McAuliffe, and Terry Yoo from the NIH.

References

1. Gobbi, D.G., Comeau, R.M., Peters, T.M.: "Ultrasound/MRI overlay with image warping for neurosurgery." MICCAI 2000, Pittsburg, PA, October 11-13: 106-114, 2000.
2. Turing, A.: "On Computable Numbers, with an application to the Entscheidungsproblem.", 1936.
3. Kohn, L.T., Corrigan, J.M., Donaldson M.S., “To Err is Human: Building a Safer Health System”, National Academy Press, Washington D.C., 2000.

4. "Guidance for Industry and FDA Staff: Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices", May 11, 2005, FDA Center for Devices and Radiological Health. http://www.fda.gov/cdrh/ode/guidance/337.html
5. "Off-the-Shelf Software Use in Medical Devices". September 9th 1999, FDA Center for Devices and Radiological Health, http://www.fda.gov/cdrh/ode/guidance/585.html
6. "Guidance for Industry: Guidance for the Submission of Premarket Notifications for Medical Image Management Devices", July 27, 2000, FDA Center for Devices and Radiological Health, http://www.fda.gov/cdrh/ode/guidance/416.html
7. "General Principles of Software Validation: Final Guidance for Industry and FDA Staff", January 11, 2002, FDA Center for Devices and Radiological Health, http://www.fda.gov/cdrh/comp/guidance/938.html
8. Ibanez, L., Schroeder, W., "The ITK Software Guide", ISBN 1-930934-10-6, 2005 http://www.itk.org/ItkSoftwareGuide.pdf
9. Schroeder, W., Martin, K., Lorensen, B., "The Visualization Toolkit, An Object Oriented Approach to 3D Graphics", Kitware Inc., 1998.
10. FLTK a cross-platform C++ GUI Toolkit, http://www.fltk.org/documentation.php/doc-1.1/toc.html
11. Lawrence Rosen "Open Source Licensing: Software Freedom and Intellectual Property Law", ISBN: 0131487876, Prentice Hall, 2004.