The Development of the Image-Guided Surgical Toolkit (IGSTK): An Open Source Package for Medical Interventions

Kevin Cleary
Patrick Cheng
Ziv Yaniv
The ISIS Center
Department of Radiology
Georgetown University Medical Center
Washington, D.C.
{cleary, cheng, yaniv}@isis.georgetown.edu

Andinet Enquobahrie
Luis Ibanez
Kitware Inc.
Clifton Park, NY
{andinet.enquobahrie, luis.ibanez}@kitware.com

Kevin Gary
Division of Computing Studies
Arizona State University
Mesa, Arizona
kgary@asu.edu

ABSTRACT

This paper describes the development of an open source toolkit for medical interventions. The problem domain is elucidated along with a brief history of the project. A description of an ongoing clinical trial using the toolkit is then provided, followed by a discussion of the toolkit architecture along with the project's emphasis on safety. Our approach is then discussed in contrast to classical approaches to real-time, safety critical systems. The paper concludes with some discussion of validation.

Keywords

Architecture Validation, Agile Methods, Safety-critical Software, Real-Time systems, State Machines

1. Problem Domain and Rationale

In the most general sense, image-guided surgery refers to the use of medical images to guide surgical decisions or minimally invasive procedures. However, the term is also used to describe an image-guided surgery system, which is a specially engineered system that provides virtual image overlay and tracking of instruments during these procedures. A typical image-guided surgery system (Figure 1) will include a 1) control computer; 2) software for image processing and display; and 3) a localizer for tracking instruments and the patient.

Figure 1. Typical computer-aided surgery system
(courtesy Medtronic Surgical Navigation Technologies)

Image-guided surgery systems were initially applied to neurosurgical procedures, since the brain is an organ where precision is critical. The brain is also surrounded by a rigid skull, which simplifies the registration task (registration involves making the correspondence between image space and localizer space). These systems incorporated optical localizers (as shown in the top left of Figure 1), which are highly accurate but depend on maintaining a line of sight between the localizer and the organ of interest.

As the image-guided field evolved, it broadened to include other organs of interest, including more deformable organs such as the liver. The recent development of electromagnetic localizers, which could track small sensor coils embedded in instruments, eliminated the link of sight tracking requirement and made it possible to track inside the body. There was a great amount of interest in these new electromagnetic trackers, and many research groups such as ours began to develop prototype systems, which were software intensive.

It was out of this environment that the image-guided surgical toolkit (IGSTK) was conceived. To enable the rapid development of new image-guided systems, the lead author proposed that an open source software package for image-guided surgery be developed. With funding from NIH, the project began in 2003, and has now developed to the point where a clinical trial based on the toolkit will begin soon, as described in the next section.

After describing the clinical trial process, this paper presents the architecture of IGSTK, first comparing and contrasting the requirements, systems architecture, and software architecture to classic approaches to real-time systems. The differences in architecture and software development process motivate the need for an architecture validation framework.

2. Clinical Trial for Biopsy

From the beginning, one of the major goals for the toolkit was to use the software in a clinical trial. Clinical trials are the standard method for evaluating the effectiveness and safety of drugs and medical devices. To conduct a clinical trial, a written document describing the trial and related issues such as safety must be written and approved by a review board.

Since biopsy is one of the most common procedures in Radiology, the Georgetown group decided to focus on lung biopsy for the initial clinical trial. It was also felt that the guidance system could potentially improve the procedure by tracking the biopsy needle and providing a continuous 3D visualization of the anatomy.

The study's purpose was to evaluate the safety and efficacy of the electromagnetic navigation system to help radiologists accurately place needles into lung lesions during CT-guided biopsies. Our hypothesis was that an electromagnetic (EM) tracking system for image-guidance can augment standard CT fluoroscopy-guided needle biopsy. To ensure the proper operation of our software and the electromagnetic tracking system, we carried out feasibility tests targeting simulated lesions in swine under an approved animal study protocol (see Figure 2 below). These tests showed that the system could work successfully in the clinical environment.

Figure 2: Swine study in CT
(GUI at top left, swine at bottom left, electromagnetic tracker on right)

Following these tests, we felt confident that the system was ready for clinical trial. The next step was to obtain IRB (institutional review board) approval and FDA IDE (investigational device exemption) approval. These approvals were obtained, and we are beginning to recruit patients (hopefully we will have results at the workshop).

The workflow of the procedure is as follows (Figure 3):

1) Patient is positioned on the CT table
2) Five skin markers are placed to serve as fiducials
3) CT scan is taken, which consists of a series of cross-sectional images
4) CT scan is exported to the image-guided system
5) Electromagnetic tracking is enabled
6) Image registration is performed: fiducials are identified in CT space and in electromagnetic space
7) Image overlay is displayed
8) Suitable path to lesion is defined (rib and critical organ interposition is avoided while minimizing the path distance to the lesion).
9) Biopsy needle is directed toward target using image overlay
10) Confirming image is obtained.
11) Biopsy sample is taken.

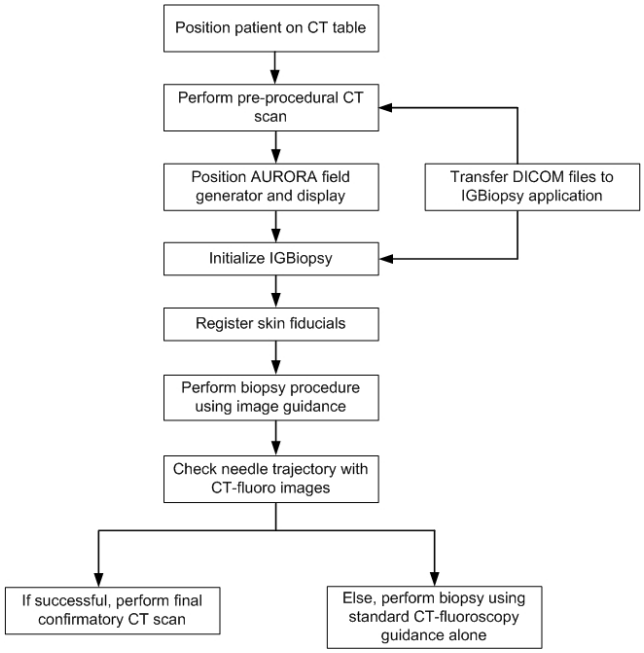

Figure 3: Workflow of procedure using electromagnetic navigation

This explanation of the clinical trial provides a backdrop for IGSTK's efforts to obtain IRB and FDA approval. Turning to the technology, we next consider how the IGSTK's architecture is derived from domain requirements.

3. IGSTK Architecture Requirements

IGSTK's main architecture requirement is *safety*. To IGSTK, safety means *patient safety*. It is important that the system is always in a responsive, known state, and that information presented to the surgeon is accurate at the given point in time. In a surgical application, a system that becomes unresponsive or reports incorrect information may have catastrophic (loss of life) consequences.

When a surgical scene is rendered on a computer display for the surgeon, it is the view of the operating room at a very recent time. Since many of the objects in the scene are in continuous movement, because they are inside of the patient or because the surgeon is controlling them, the accuracy of the position is related to the consistency of

time for each object. In other words, when the graphic display shows where the surgical needle was located at 9:06 am, it should appear along with the location of the patient's liver at 9:06AM. IGSTK takes measures for preventing the accidental display of the position of the needle where it was at 9:06AM along with the patient's liver where it was at 8:54AM. Synchronicity of the objects in the scene is extremely important because it is from their relative position that the surgeon will derive the most useful information for proceeding with the intervention.

Similarly, when a surgical system is in use, the system cannot "lock up" or become unresponsive without some notification to the surgeon and a fail-safe mechanism that allows the technology to be removed from the environment. IGSTK does not control physical objects whose malfunction would cause loss of life. However, an unresponsive surgical application may result in a misinterpretation of a surgical scene if indicators are not present to notify the surgeon. Furthermore, fault detection within the system must be handled in such a way that ensures the current state of the system is (at all times) known. IGSTK implements a variation of the State Pattern [1] using state machines to ensure operations are only performed when the system is in a known state which permits the operation.

4. IGSTK Architecture

Given IGSTK's requirements one would expect it is a candidate for a classic real-time systems approach. However, IGSTK's approach differs somewhat from this expectation. We describe IGSTK's approach in contrast to classic approaches to real-time, safety-critical systems.

4.1 IGSTK Systems Architecture

From a hardware perspective, an image-guided surgical system supported by IGSTK includes the surgical environment (often outfitted with imaging devices), a computer, tracking devices, and tracker tools. Tracker tools are physical objects one holds in her/his hand, such as a surgical instrument. These instruments have attached fiducials or markers that allow their position to be read by the tracking device via a vendor-defined interface. The tracker device itself is connected to a computer via a standard interface. The nature of the connector is not a hard constraint in IGSTK; RS232 serial, USB, or firewire may be used provided the component interface encapsulates the communication interface.

The tracker device has its own processor and a given frame rate which determines how often it reads the position of its attached tools. Typical frame rates may vary between 15 Hz and 60 Hz. IGSTK currently supports the NDI POLARIS optical trackers, the NDI AURORA magnetic trackers, the Claron Micron tracker, and the Ascension Flock of Birds magnetic trackers. The IGSTK team is currently working on integrating other tracker devices and new hardware components such as video frame grabbing boards.

IGSTK considers the computational power of the computer to be an application-level requirement. While there are minimal requirements such as the ability to render images, perform complex mathematical operations, and store image data in memory, these do not go beyond the capabilities of a low-cost off-the-shelf (OTS) workstation. If a particular application requires specialized processing or additional displays, it is up to the systems application developer to determine the computational resources required.

4.2 IGSTK Software Architecture

IGSTK employs the component-based layered software architecture shown in Figure 4. Requests from an IGSTK-enabled application enter the View layer on the left. From there requests are made left-to-right across layers, and responses are passed back right-to-left.

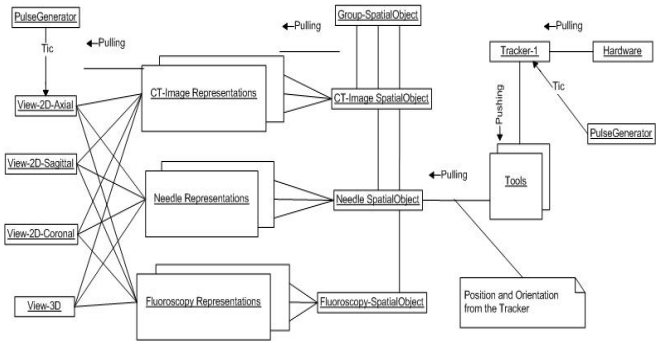

Figure 4. IGSTK component-based layered architecture

View objects are responsible for rendering representations of physical objects on the computer display. In Figure 4, the existence of four view objects suggests an application with a four-quadrant display. The Representation layer provides a canonical representation of spatial object that can then be mapped on to multiple views. Spatial objects are the internal representation of physical objects in visible and invisible space (inside a patient's anatomy), and the Tracker and Tools are software components representing these physical devices.

Communication between components in different layers uses a request-response paradigm. A component in one layer requests a service of a component in an adjacent layer. The service provider passes a response back (with or without a data payload) via events; requestors must first register interest in receiving the response event. This pattern is shown in Figure 5.

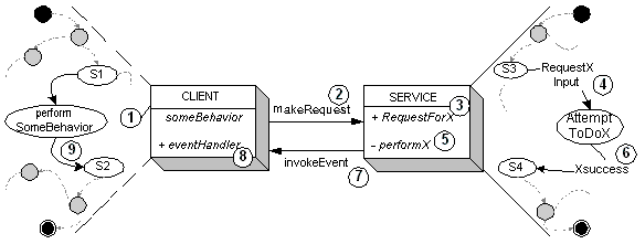

Figure 5. IGSTK component interaction pattern

In Figure 5, CLIENT is performing some behavior while in a well-known state (1). CLIENT makes a request (2) via a public method invocation on SERVICE. SERVICE accepts the request (3) and translates it to an input to an internal state machine (4). Based on the transition, a private method invocation on component SERVICE also can occur (5). Upon successful completion of the computation (6), SERVICE generates an event (7) and dispatches the event to CLIENT (8). CLIENT translates the event into an input to its state machine (9), thereby avoiding error-prone conditional logic on return values.

This call sequence that governs a component behavior in IGSTK takes place within timing constraints. IGSTK provides two mechanisms, a Pulse Generator object and a pseudo Real-Time Clock (RTC) to enforce timing and synchronization. The Pulse Generator generates discrete ticks at an application-specified interval. Pulse Generator objects are employed for the main application thread (upper left Figure 4) and the tracker's internal thread (right side Figure 4), and set to the appropriate refresh rates for the application's view and tracker components. The refresh rates of View classes must be selected according to the capabilities of the graphics hardware available, the complexity of the scene to be rendered, and the actual rate of interaction required by the clinician. The refresh rate of the Tracker classes must be selected according to the actual tracker hardware rate of refreshing position information, the rate of transmission set on the communication layer, for instance the baud rate of the serial link, and the purpose of displaying the tracked object in the surgical scene.

The left-to-right request pattern of Figure 4 means that the main application thread determines the responsiveness of the application as a whole; the user sees refreshes at the rate of View's Pulse Generator. Typically (though not always) the Tracker's Pulse Generator will operate at a higher frequency, yet it merely writes last know transforms to a buffer area, whose data is then pulled from a request in the main application thread. This is the only situation where thread safety comes into play.

The RTC mechanism is responsible for assigning time intervals for the validity of tracker data (matrix transforms). Timestamps on transforms are a time-to-live (TTL) mechanism that lets downstream IGSTK components know when transforms should not be used in object position calculations, and when to expect a new transform. If the transform of an object has expired by the time it needs to render its appearance on the display, then it can decide not to show that particular instrument, or to flag it in blinking mode, or in a special color. The purpose of this change in representation is to warn the surgeon that the current position of that object is not known at this point. Such an event may be the consequence of someone blocking the line of sight of an optical tracker, or an accidental disconnect of a physical tracking tool. In either case, it is very important to let the surgeon know that the position of that particular object in the display cannot be trusted.

The application of internal state machines, request-response pattern using events, and timing mechanisms may lead one to believe the IGSTK software architecture has reactive, concurrent, and asynchronous characteristics, but (for the most part) this is not the case. IGSTK applications make synchronous requests within a main application thread. Software components are not required to be thread safe nor mapped onto a separate processor such as is typical in a distributed systems environment. Although the tracker is a notable exception, even its processing is bounded within the IGSTK framework through the software component's interface to the rest of the software architecture.

4.3 Discussion: IGSTK Architecture

At first glance, IGSTK's systems and software architecture would appear to follow several commonly found design principles and practices in real-time systems. A closer consideration of the architecture's guiding requirements and the key design decisions with respect to those requirements reveals this is not the case.

To clarify IGSTK's approach, it is worthwhile to consider if applications constructed on top of the IGSTK platform are real-time systems at all. Many definitions of real-time systems abound; we consider the definition widely referenced by Stankovic [2] who says that in a real-time system the "correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced." This is certainly the case in IGSTK, as the position of objects reported by the tracker device must correspond in time with the rendering of the corresponding objects in the computer application. If timing constraints are not met in IGSTK, information is lost. However, losing such information is not catastrophic from an application perspective, as IGSTK will not report incorrect information at any time. As described above, a visual indicator may be presented to the surgeon when valid information is not available to render the surgical scene so the result is not catastrophic. Therefore, according to the above definition, IGSTK applications are in the class of real-time systems.

The architectures of real-time systems, however, are typically constructed on design principles that have evolved over the past two-plus decades. For example, a real-time operating system (RTOS) is often employed to help guarantee scheduling and other timing considerations. IGSTK supports several OTS operating systems, including various flavors of Windows and UNIX. Hard real-time systems typically have specialized (embedded) processing capabilities. IGSTK integrates with tracker devices which have embedded processing capabilities, though the computer it is attached to can be an OTS workstation. Hard real-time systems typically use multiple distributed processors and/or multithreading to facilitate throughput and guarantee performance with respect to timing constraints. IGSTK currently assumes a non-distributed computing environment employing only a single application thread (though additional threads are used within the bounds of the tracker component and services such as logging). The result is that IGSTK is not a *reactive, concurrent* system in the classic sense, but instead is able to make simplifying timing assumptions in the name of safety.

Returning to the principle architectural requirement of safety, IGSTK also takes a different approach to address this requirement than most safety-critical systems. Leveson [3] provides the accepted definition for safety as "freedom from accidents", where an accident is "an undesired and unplanned event that results in a loss (including loss of human life or injury, property damage, environmental pollution, etc.)." IGSTK takes the view that accidents, while undesirable, are not unavoidable, particularly in an environment under few restrictions. Recall IGSTK is an open source platform intended to be used on OTS hardware and operating system environments; as such there are too many variables not under its control. To state another way, IGSTK does not assume underlying services provide guaranteed quality of service with respect to reliability.

IGSTK instead takes the approach that although an "undesired and unplanned event" may happen, IGSTK should always maintain a known state in response to the event. IGSTK uses an implementation of the State Pattern based on internally referenced state machines to maintain a set of known states of its components and services at any given instant in time. Even in situations where system faults occur, from a software perspective IGSTK maintains consistent state, and may only enact appropriate behaviors (including the reporting of data to the application or the rendering of the surgical scene) in that state.

We have already suggested examples of such faults – a person obstructing the line of sight of an optical tracker; or the sudden disconnection of a tracker tool. In either situation, the IGSTK components transition to appropriate states that restrict the set of behaviors (services) an application may request. In these examples, an application will no longer be able to request the position of an instrumented surgical tool during this period.

To summarize, as a software platform IGSTK employs the State Pattern through internal state machines to provide *software safety* in the face of an environment that may not guarantee a high degree of *system reliability*. IGSTK takes this approach as part of its mission to be an open source, portable platform that supports both surgical applications and research endeavors. A surgical system must take additional measures to ensure *system safety* and application-defined levels of reliability.

5. IGSTK Architecture Validation

Section 4 describes IGSTK's approach to architectural requirements at the system and software level, and contrasts these with accepted approaches in real-time and safety-critical systems. IGSTK's development process also differs from approaches typically used in real-time and safety-critical systems. IGSTK's employs an *Agile* methodology[1], which favors continuous builds and testing over extensive design activities.

For example, consider the "classic" approach when basing an architecture on state machines. Typically, one would create a model of the state machine in a tool or formal language. Then the model would be verified through analytical (provable) means or extensive simulation. The model would then be used to generate code through a correctness-preserving automated or manual process.

IGSTK provides its own internal state machine execution semantics. State machines are coded "by hand", without the aid of visual modeling tools, formal languages, or OTS runtime support. The ability to completely control execution semantics, and not introduce a 3rd party dependence is viewed as a safety benefit to IGSTK. It means IGSTK is not tied to OTS technologies, and does not have to map IGSTK execution semantics onto tools whose semantics generally were intended for embedded systems with asynchronous, multithreaded, (and perhaps distributed, concurrent) runtime environments.

Agile methods and a lack of a formal specification of the architecture are an unusual combination for real-time safety-critical software, although it is not unprecedented within the medical device community [6]. A particular concern is that although IGSTK's continuous testing process emphasizes both *black box* (functional) *white box* (structural) testing at the component level, it does not validate state machine structure (static analysis) or expected runtime properties (dynamic analysis). An IGSTK component that exhibits proper functional behavior with respect to unit tests may have an undiscovered defect in the

[1] Space limitations preclude a detailed description of IGSTK's Agile process. The interested reader is referred to [4] or [5].

construction of its state machine. Further, there is currently no means by which the global state of the system, taken as the composition of states of its instantiated components, is validated for consistency over time.

To address these issues, we are currently working on architecture validation post-construction. We extract a model of component state machines from the source code, and then validate this model with respect to architecture characteristics relevant to IGSTK. Our current areas of emphasis are on static (structural) analysis and global state consistency through dynamic analysis (runtime constraint checks). These techniques are being integrated into the continuous build and testing Agile process to ensure these checks are performed as often as the traditional unit tests.

For static analysis, we consider domain independent and domain dependent situations. Domain independent analysis includes checking for determinism, and performing state machine variants of node, edge, and path coverage through simulation. Domain dependent analysis includes checking that all state machines conform to IGSTK-specific design principles – for example, ensuring all component state machines respond to all possible inputs at all times.

Dynamic analysis is entirely domain dependent; the constraints placed on the valid states in a composed state machine[2] depend on the nature of the components. For example, a Spatial Object instance should not report a location if the Tracker is in an "OFF" state and unable to report the actual location of the physical object.

IGSTK's informal architecture approach and reliance on Agile process principles (particularly the continuous build and test practices) create unique constraints on our validation efforts. Reorienting toward a full architecture specification (with review), model verification, and code generation with underlying runtime support is not an option. The validation toolset has to validate state machine structure and execution and report its results as part of the Agile process. Our challenge is to create a toolset that not only addresses the unique approach to state machine design and implementation in IGSTK, but to integrate these validation tools into an Agile development process that relies heavily on automated tool support.

6. Summary

This paper presented the Image-Guided Surgical Toolkit (IGSTK), focusing on the project motivation and derived architectural requirements. IGSTK is an open source, open platform software toolkit for practitioners and researchers working on technology supported surgical applications. IGSTK applications fall under the category of real-time applications with safety-critical requirements. However, several distinctions in IGSTK's architecture and process that differ from more traditional approaches in these areas were discussed.

As a software toolkit, IGSTK is limited in the assumptions it can make at the system level. This creates an additional burden on the software to be, in a sense, *self-aware*. IGSTK software is always in a known state with predictable behaviors given that state in the face of a potentially unreliable system environment.

As a toolkit developed through Agile software development methods, IGSTK's approach does not include rigorous design activities. The lack of a verifiable model and the emphasis on continuous testing forced us to devise a collection of architectural validation practices that can be incorporated into the existing process.

There have been two major releases of IGSTK and it is in active development for future releases. A clinical trial is in process that will hopefully yield positive results in the near future. Interested readers may visit the community website at http://www.igstk.org

7. References

1. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading MA, 1995.
2. Stankovic, J.A., "Misconceptions about Real-time Computing: A Serious Problem for Next-Generation Systems" *IEEE Computer*, 21(10):10-19, October 1988.
3. Leveson, N. *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA, 1995.
4. Cleary, K., Ibanez, L., Gobbi, D., Cheng, P., Gary, K. Aylward, S., Jomier, J., Enquobahrie, A., Zhang, H., Kim, HS., and Blake, M.B. *IGSTK: The Book*. Self-published Manuscript, The ISIS Center, Georgetown University, 2007.
5. Gary, K., Ibanez, L., Aylward, S. Gobbi, D., Blake, M.B., and Cleary, K. IGSTK: An Open Source Software Toolkit for Image-Guided Surgery. *IEEE Computer*, 39(4):46-53, April 2006.
6. Spence, J.W., "There has to be a better way! [software development]" *Proceedings of the Agile Conference*, July 2005.
7. Harel, D. "Statecharts: A Visual Formalism for Complex Systems" *Science of Computer Programming 8*, North Holland Publishing Company, the Netherlands, 1987.

Acknowledgments. This project is a collaboration between Georgetown University, Kitware Inc., Arizona State University, SINTEF Norway, and Atamai Inc. This work was funded by NIBIB/NIH grant R01 EB007195. Additional support was provided by U.S. Army grant W81XWH-04-1-007. We thank our other collaborators throughout the project, including David Gobbi of Atamai Inc. and Frank Lindseth, Geir Arne Tangen, Ole Vegard Solberg, Arild Wollf, Torleif Sandnes of SINTEF Health Research, Medical Technology (and the National Centre for 3D Ultrasound in Surgery), Trondheim, Norway

[2] The reader may suspect that "composed state machines" with "global state" are equivalent to Harel [7] hierarchical statecharts, and we suspect this is indeed the case.