

# Surgical Assistant Workstation Software Architecture Document

Johns Hopkins University and Intuitive Surgical Inc.

Version 1.4

Date: 2007/08/09 20:19:14

## Revision History

| <b>Date</b> | <b>Version</b> | <b>Description</b>                                   | <b>Author</b> |
|-------------|----------------|--|---------------|
| 1/3/07      | 1.0            | Initial Version                                      | PK            |
| 5/20/07     | 1.1            | Revisions to reflect recent architecture discussions | SPD           |
| 5/28/07     | 1.2            | Added definitions and communication primitives       | PK            |
| 6/13/07     | 1.3            | Rewritten using $\LaTeX$                             | PK            |
| 6/21/07     | 1.4            | Updated version for internal review                  | PK/SPD        |

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>3</b>  |
| 1.1      | Purpose . . . . .  | 3         |
| 1.2      | Scope . . . . .  | 3         |
| 1.3      | Definitions, Acronyms, and Abbreviations . . . . .                         | 3         |
| 1.4      | References . . . . .   | 5         |
| 1.5      | Methodology . . . . .  | 7         |
| <b>2</b> | <b>Architecture Overview</b>   | <b>7</b>  |
| <b>3</b> | <b>Use Case View</b>   | <b>9</b>  |
| 3.1      | Overview . . . . .   | 9         |
| 3.2      | Image Guidance: da Vinci with Laparoscopic Ultrasound Instrument . . . . . | 9         |
| 3.3      | Image Guidance: da Vinci with Medical Image Overlay . . . . .              | 12        |
| 3.4      | Mentoring . . . . .  | 13        |
| 3.5      | Haptic Guidance: Virtual Fixtures . . . . .                                | 15        |
| 3.6      | Research Hardware: Snake Robot . . . . .                                   | 16        |
| <b>4</b> | <b>Logical View</b>  | <b>16</b> |
| 4.1      | Overview . . . . .   | 16        |
| 4.2      | Communication Primitives . . . . .   | 17        |
| 4.3      | Logical Components . . . . .   | 19        |
| <b>5</b> | <b>Process View</b>  | <b>24</b> |
| 5.1      | Overview . . . . .   | 24        |
| 5.2      | Process Components . . . . .   | 24        |
| <b>6</b> | <b>Deployment (Physical) View</b>  | <b>27</b> |
| 6.1      | Overview . . . . .   | 27        |
| 6.2      | Physical Components . . . . .  | 27        |
| <b>7</b> | <b>Implementation (Development) View</b>                                   | <b>28</b> |
| 7.1      | Overview . . . . .   | 28        |
| 7.2      | Development Components . . . . .   | 28        |

# 1 Introduction

## 1.1 Purpose

This document provides a comprehensive architectural overview of the Surgical Assistant Workstation for Teleoperated Surgical Robots (SAWTSR or SAW), using a number of different architectural views to depict different aspects of the system design. It is intended to capture and convey the significant architectural decisions that have been made, based on the scope and requirements of the SAW concept, as defined in “*System Requirements for the Surgical Assistant Workstation*,” Rev 2, January 29, 2007.

## 1.2 Scope

The Surgical Assistant Workstation will provide a powerful research platform to aid the development of new methods for enhancing the capabilities of robot-assisted laparoscopic surgery—as presented by the da Vinci telerobotic system today—by providing fully-integrated image guidance, and data-enhanced intra-operative assistance to the surgical team and to the surgeon in particular. The core piece of this effort is a modular, open-source software application framework that is designed to combine existing research infrastructure at Johns Hopkins University (JHU) with the clinical systems at Intuitive Surgical Inc. (ISI) in order to support new development and enhancement of the da Vinci platform and similar robot systems.

*This document describes the architecture of the Surgical Assistant Workstation application framework, which can have many different physical implementations, depending on each specific application and its associated hardware components (i.e., sensors, vision systems, imaging systems and robots) and software algorithms, as defined by the application developer.*

## 1.3 Definitions, Acronyms, and Abbreviations

### 1.3.1 Definitions

**Callback:** A callback is an function (e.g., method) that is defined in one context (object) and called from another. In a multi-threading scenario, it is necessary to consider mutual exclusion issues because the callback will usually be invoked from a different thread. In a network (or multi-process) scenario, the callback must be implemented using some form of a proxy (e.g., middleware) because in general it is not possible to have one process call methods in another process.

**Command:** A message that is sent from one object to another (see 4.2.1). Typically, a command is issued by a “higher level” object and the recipient is a “lower level” object. An **Event Object** is a command that originates

in a “lower level” object and is propagated to one or more “higher-level” objects.

- Device Driver:** Hardware- and operating-system-specific code that interfaces to a physical device, usually runs at a protected (kernel) level. May include interrupt handlers to service interrupts from the device. Device drivers are not within the scope of the CISST software.
- Device Interface:** CISST software object that encapsulates a physical device and any associated **Device Driver**. A device interface may include a thread of execution (see **Device Task**) or may just be a “wrapper” around the device driver or vendor API (see **Device Wrapper**).
- Device Task:** A **Device Interface** that is implemented as a **Task** (i.e., includes its own thread of execution). There is no functional difference between a **Device Task** and a **Task**, but the former is defined to clarify the specialization of the task.
- Device Wrapper:** A **Device Interface** that does not include its own thread of execution.
- Event:** An event is sent from one object to another to indicate an “exceptional” condition (see 4.2.3). For generality, an event will be implemented using a callback mechanism, where the callback function will accept an **Event Object** as a parameter. Typically, an event is sent from a “lower level” object to one or more “higher level” objects.
- Event Object:** A type of **Command** object that is used to specify the data associated with an “exceptional” condition. An **Event Object** originates in a “lower level” object and is propagated to one or more “higher-level” objects. It will contain some fixed fields, such as originating process identification and timestamp, and some variable fields (the payload) that is event-specific.
- Mailbox:** A **Queue** for incoming messages to a **Task**. Typically, we assume a FIFO (first-in-first-out) implementation, but that is not a requirement.
- Queue:** A queue is a data structure that buffers data produced by one object (e.g., **Task**), and makes it available to another object.
- State Data Table:** A two-dimensional table that contains a time history of all important data (i.e., “State”) for a **Task**. This table is indexed by “time” and by “data id”. Essentially, it is a (time-indexed) **Queue** of data vectors that is most often accessed as a LIFO (last-in-first-out) in order to get the most recent data.
- State Read:** A state read occurs when a “higher level” object wishes to access data (state) from a “lower level” object (see 4.2.2). If the lower level object is a **Task**, this is accomplished by reading from the **State Data Table**.
- Task:** CISST software object that is derived from **Device Interface**, and

includes a separate thread of execution. A task maintains a **State Data Table** and a **Mailbox** for incoming commands.

**Telestration:** Freehand sketching over a motion picture image. Used in the da Vinci system to "tele-illustrate" on the endoscopic video image from an external touch screen.

### 1.3.2 Acronyms

|               |   |
|---------------|---|
| <b>DICOM</b>  | Digital Imaging and Communications in Medicine                  |
| <b>DVI</b>    | Digital Visual Interface  |
| <b>ECM</b>    | Endoscopic Control Manipulator                                  |
| <b>FIFO</b>   | First-in-first-out data access policy                           |
| <b>FRVF</b>   | Forbidden region virtual fixtures                               |
| <b>GVF</b>    | Guidance virtual fixtures                                       |
| <b>IRE</b>    | Interactive Research Environment                                |
| <b>LapUS</b>  | Laparoscopic Ultrasound   |
| <b>LIFO</b>   | Last-in-first-out data access policy                            |
| <b>MaM</b>    | Masters as Mice   |
| <b>MRML</b>   | Medical Reality Markup Language                                 |
| <b>MTM</b>    | Master Tool Manipulator   |
| <b>PACS</b>   | Picture Archiving and Communication System                      |
| <b>PSM</b>    | Patient Side Manipulator  |
| <b>SAW</b>    | Surgical Assistant Workstation                                  |
| <b>SAWTSR</b> | Surgical Assistant Workstation for Teleoperated Surgical Robots |
| <b>SDI</b>    | Serial Digital Interface  |

## 1.4 References

### Project-specific:

1. P. Kruchten, "Architectural Blueprints-The '4+1' View Model of Software Architecture," *IEEE Software*,12(6), pp. 42-50, 1995.
2. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, *IEEE Std 1471-2000*.
3. "System Requirements for the Surgical Assistant Workstation," Rev 2, January 29, 2007

4. "Development of a Surgical Assistant Workstation for Teleoperated Surgical Robots," NSF proposal number 0646678, August 2006.
5. "Ultrasound Guidance for a Laparoscopic Surgical Robot," NIH SBIR Phase II proposal, R42-RR019159.
6. "Intuitive Surgical daVinci API v5.0 Reference Manual", generated July 14, 2006.
7. J. Leven, D. Burschka, R. Kumar, G. Zhang, S. J. Blumenkranz, X. Dai, M. Awad, G. Hager, M. Marohn, M. Choti, C. Hasser and R. H. Taylor "DaVinci Canvas: A Telerobotic Surgical System with Integrated, Robot-Assisted, Laparoscopic Ultrasound Capability," in MICCAI, vol. LNCS 3749, J. Duncan and G. Gerig, Eds. Palm Springs, CA: Springer-Verlag, 2005, pp. 811-818.
8. J. Leven, "A Telerobotic Surgical Systems with Integrated Robot-Assisted Laparoscopic Ultrasound Capability", MS Thesis, Computer Science, Johns Hopkins University, Baltimore, 2005.

#### **Calibration and Registration Techniques:**

1. B.K.P. Horn, "Closed-form solution of absolute orientation using unit quaternions," J. Opt. Soc. Amer. A, Vol. 4, No. 4, pp. 629-642, Apr. 1987.
2. K.S. Arun, T.S. Huang, S.D. Blostein, "Least-Squares Fitting of Two 3D Point Sets", IEEE PAMI, Vol. 9, No. 4, pp. 698-700, Sept. 1987.
3. S. Umeyama, "Least-Squares Estimation of Transformation Parameters Between Two Point Patterns", IEEE PAMI, Vol. 13, No. 4, pp. 376-380, Apr. 1991.
4. E. Boctor, A. Viswanathan, M. Choti, R. Taylor, G. Fichtinger, G. Hager, "A Novel Closed Form Solution for Ultrasound Calibration," IEEE Intl. Symp. Bio. Imag. (ISBI), Arlington, VA, pp 527-530, Apr. 2004.
5. P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, pp. 239-256, Feb. 1992.

#### **Virtual Fixtures and Constrained Optimization:**

1. A. Kapoor, M. Li, R.H. Taylor, "Constrained Control for Surgical Assistant Robots", Proc. IEEE Intl. Conf. on Robotics and Automation, Orlando, FL, May 2006, pp 231-236.
2. M. Li, A. Kapoor and R. H. Taylor "A Constrained Optimization Approach to Virtual Fixtures," in IROS. Edmonton, Alberta, Canada, 2005.

3. M. Li and R. H. Taylor, "Performance of Teleoperated and cooperatively controlled surgical robots with automatically generated spatial virtual fixtures.," in IEEE International Conference on Robotics and Automation. Barcelona, Spain, 2005.
4. M. Li, "Intelligent Robotic Surgical Assistance for Sinus Surgery", Ph.D. Thesis, Computer Science, The Johns Hopkins University, Baltimore, Maryland, 2005.
5. M. Li and R. H. Taylor, "Spatial Motion Constraints in Medical Robots Using Virtual Fixtures Generated by Anatomy," in IEEE Conf. on Robotics and Automation. New Orleans, 2004, pp. 1270-1275.

## 1.5 Methodology

In this document, the software architecture is expressed using the 4+1 views methodology (Kruchten 1995), where the 4 views are: Logical, Process, Deployment (Physical), and Development (Implementation); the +1 view is Scenarios (Use Cases).

## 2 Architecture Overview

Our goal is to create a unified assistive environment for surgery that integrates robotic devices, preoperative and intraoperative data sets, surgical task models, and human-machine cooperative manipulation, as shown in Figure 1. The following major sub-systems are present within the SAW framework:

- **Robot API:** this subsystem provides a general interface to the master and patient-side manipulators. The design is based on the existing daVinci research API, but is intended to accommodate other robot systems, including research systems.
- **Collaborative Robot Control System:** this subsystem facilitates teleoperation control between master-slave pairs, as well as collaborative control of multiple master manipulators and/or patient-side slave manipulators.
- **Video processing:** this subsystem provides mechanisms for acquiring and processing streams of images, including 2D ultrasound and stereo endoscopic video. Such image processing pipelines can be used to implement tool and tissue tracking algorithms.
- **Tool tracking:** a specialized image processing pipeline is defined for tracking the positions of surgical instruments using a combination of kinematic and stereo vision feedback.
- **Calibration and registration:** this subsystem provides software tools for determining device calibration, as well as methods for computing coordinate transformations between data sources (i.e., registration). Such tools include kinematic calibration, camera calibration, ultrasound calibration, pre-operative and intra-operative image registration, video registration and overlay, etc.

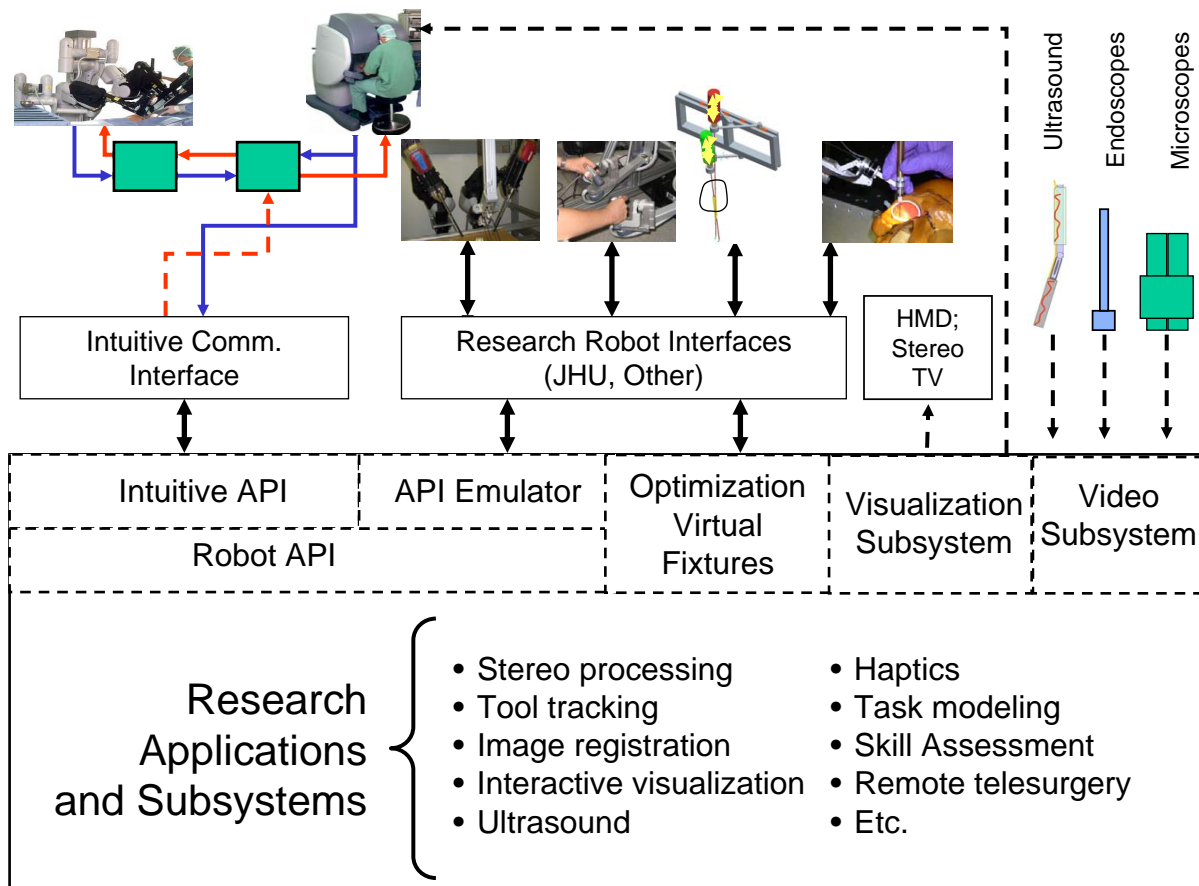


Figure 1: Overview of the SAWTSR architecture.

- **Other device interfaces:** this subsystem provides interfaces to devices other than robots and video systems, including force sensors, foot pedals, tissue oxygenation sensors, etc.
- **User Interface:** the user interface facilitates interactive manipulation and visualization of 2D and 3D data objects, including medical images and video, directly within the surgical console. A 3D graphical user interface manages user interaction from various input devices (including the master manipulators of the da Vinci MTMs) and renders a menu system and graphical overlays to the stereo display of the surgical console. A 3D Brick Manager (as opposed to a 2D Window Manager) provides application-level “widgets” and interaction logic. A secondary user interface—called the Staff Console—will be provided to support the surgical interface. This is a conventional 2D interface that is intended for planning and monitoring outside of the surgical console.
- **Data Management:** this subsystem provides means to both import and export archived application data, including medical images, models, surgical plans and an-

notations. In its implementation, this sub-system could accommodate data in various formats, including Medical Reality Markup Language (MRML), DICOM and clinical PACS.

- **Logging and Monitoring:** performance monitoring, state logging and recovery.
- **SAW application framework:** this application development framework integrates all of the sub-systems described above. It is modular by design, such that it can be used to implement different physical architectures and applications. It provides the main application context and event loops, as well as communication mechanisms for interconnecting modules and devices with the application logic.

The functional and performance requirements of each of these core sub-systems are detailed in the system requirements document, entitled: “System Requirements for the Surgical Assistant Workstation,” Rev 2, January 29, 2007.

## 3 Use Case View

### 3.1 Overview

This section describes a selection of specific use cases in the form of task-level workflows, in order to provide context and motivation for the architecture design that follows.

- Image Guidance: laparoscopic ultrasound
- Image Guidance: medical image overlay
- Mentoring: supervisory and trainee consoles
- Haptic Guidance: virtual fixtures
- Research Hardware: snake robot

### 3.2 Image Guidance: da Vinci with Laparoscopic Ultrasound Instrument

**Brief Description:** Dynamic overlay of a dynamic laparoscopic ultrasound image on a tracked LapUS instrument in the stereo endoscope view of the da Vinci system. This is a simplified use case for illustrative purposes only.

- Pre-conditions:**
- A laparoscopic ultrasound instrument is attached to one of the active PSMs and is inserted through a cannula into the body cavity.
  - The ultrasound transducer has been calibrated to the da Vinci instrument.

- Endoscopic video outputs from the da Vinci system are connected to the SAW.
- Video output from a diagnostic ultrasound device is connected to the SAW.
- Video output of the SAW is connected to the da Vinci master console.
- SAW is connected to the da Vinci system via Ethernet.
- The surgeon operates the master console. While the master clutch pedal is not pressed, the stereo display shows the live endoscopic images and operates normally.

**Trigger:** The surgeon depresses the master clutch pedal on the da Vinci surgical console.

- Basic Flow:**
1. The surgeon closes and then releases both MTM grips in order to enter *Masters-as-Mice mode*.
  2. GUI mode becomes active, a 3D pointer/cursor and menu system are overlaid onto the surgical console. Graphical tool icons appear at the tip of each PSM instrument.
  3. The surgeon moves the 3D pointer within his/her field of view by manipulating one of the MTMs (the primary).
  4. The surgeon moves the 3D pointer over the tool icon attached to the ultrasound instrument.
  5. The surgeon closes the grip on the primary MTM to signal click/select.
  6. The pull-down menu opens to display two options, namely: “LapUS Flashlight View” and “LapUS Inset View”.
  7. The surgeon moves the primary MTM to highlight “LapUS Flashlight View”.
  8. The surgeon releases the grip on the primary MTM.
  9. The ultrasound image plane is overlaid onto the ultrasound instrument.
  10. The surgeon releases the master clutch.
  11. The menu system and tool icons disappear, while the ultrasound overlay remains.
  12. The surgeon closes the MTM grips to re-enter *following mode*.
  13. The ultrasound overlay moves with the LapUS instrument, rigidly fixed to the coordinate frame of the ultrasound transducer.

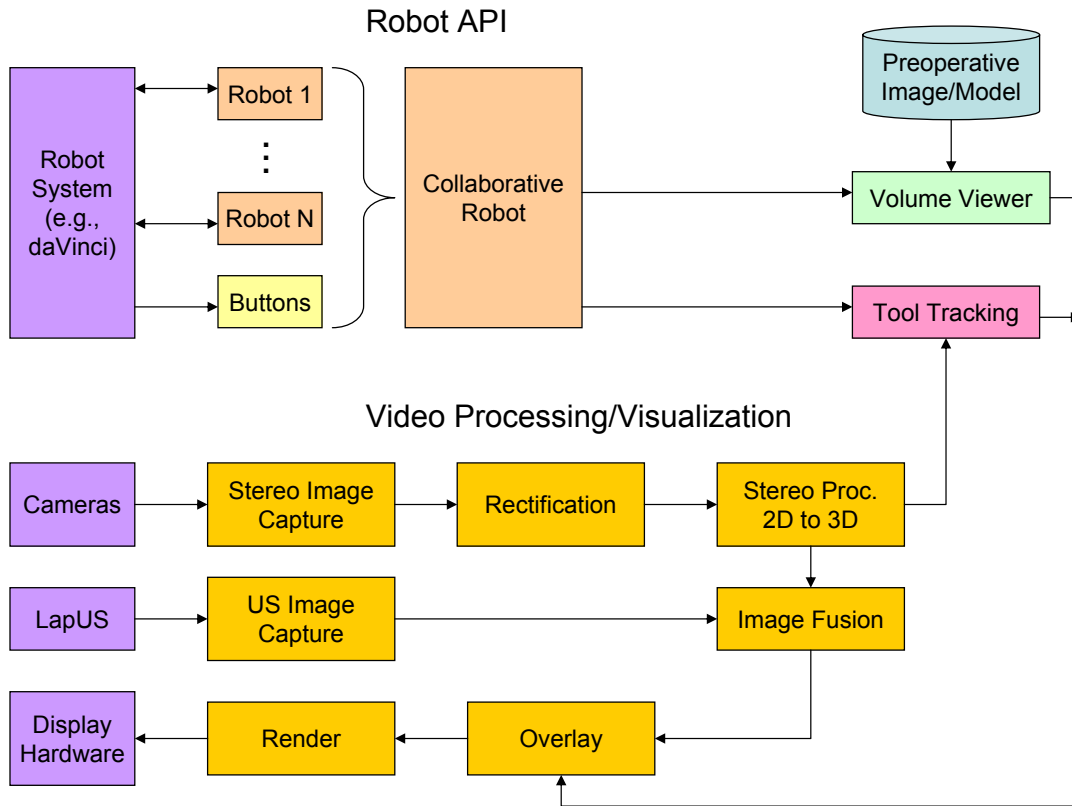


Figure 2: Illustrative data flow

**Post-conditions:**

- If MaM mode was not activated, then the system operates normally (no GUI appears).
- If MaM mode was activated, the system operates as in the case of returning from clutch mode.
- The ultrasound image overlay remains on display in the console and tracks the motion of the LapUS instrument.

**Data Flow:**

Figure 2 shows an illustrative data flow diagram for this use case. It focuses on the robot API and the pipeline for video processing and visualization. This figure also shows the tool tracking and volume viewer subsystems. Although not specifically shown, calibration and registration functions are required.

### 3.3 Image Guidance: da Vinci with Medical Image Overlay

**Brief Description:** Overlay of a medical image volume within the surgical console of a da Vinci system. This is a simplified use case for illustrative purposes only.

**Pre-conditions:**

- Endoscopic video outputs from the da Vinci system are connected to the SAW.
- Video output of the SAW is connected to the da Vinci master console.
- SAW is connected to the da Vinci system via Ethernet.
- The surgeon operates the master console. While the master clutch pedal is not pressed, the stereo display shows the live endoscopic images and operates normally.

**Trigger:** The surgeon depresses the master clutch pedal on the da Vinci surgical console.

**Basic Flow:**

1. The surgeon closes and then releases both MTM grips in order to enter *Masters-as-Mice mode*.
2. GUI mode becomes active, a 3D pointer/cursor and menu system are overlaid onto the surgical console. Graphical tool icons appear at each of the PSM tools.
3. The surgeon moves the 3D pointer within his/her field of view by manipulating one of the MTMs (the primary).
4. The surgeon moves the 3D pointer over a pull-down menu button visible on the overlaid menu system. The button is marked: "View Image Volume".
5. The surgeon closes the grip on the primary MTM to signal click/select.
6. The pull-down menu opens to display a list of image data sets by their unique identifiers (predefined).
7. The surgeon moves the primary MTM to highlight the desired image volume for viewing.
8. The surgeon releases the grip on the primary MTM.
9. An annotated image volume bounding box appears in the console view. A single image slice is shown within the bounding box. Crosshairs indicate the location of the origin and eight corners of the image volume, as well as the four corners of the active slice plane.

10. The surgeon moves the 3D pointer over the crosshair at the origin of the image volume and closes the grip of the primary MTM. The primary and secondary MTMs are used to pan, zoom and rotate the image volume object:
  - (a) PAN: the primary MTM translates the origin of the image volume.
  - (b) ZOOM: the secondary MTM is selected (by closing its grip); distance between primary and secondary MTMs controls zoom level.
  - (c) ROTATE: the secondary MTM is selected and positioned over one corner of the image volume; relative motion between primary and secondary MTMs controls volume orientation.
11. The surgeon releases both MTM grips.
12. The surgeon moves the 3D pointer over one of the corners of the slice plane and closes the grip of the primary MTM. The primary and second MTMs are used to reformat the slice plane:
  - (a) TRANSLATE: the slice plane follows the motion of the primary MTM.
  - (b) ROTATE: the secondary MTM is selected and positioned over a second corner of the slice plane; relative motion of the primary and secondary MTMs controls slice plane orientation.
13. The surgeon releases the master clutch.
14. The menu system and image volume overlay are minimized. The surgeon closes the MTM grips to re-enter *following mode*.

**Post-conditions:**

- If MaM mode was not activated, then the system operates normally (no GUI appears).
- If MaM mode was activated, the system operates as in the case of returning from clutch mode.

**Data Flow:**

Figure 2 shows an illustrative data flow for elements relevant to this use case.

### 3.4 Mentoring

**Brief Description:** Two da Vinci surgeon consoles are coupled with a single patient-side cart for mentoring purposes. One surgeon console is designated the supervisor console, while the second is the trainee console. This is a simplified use case for illustrative purposes only.

**Pre-conditions:**

- Two da Vinci surgeon consoles are interfaced with the SAW.

- One da Vinci patient-side cart is interfaced with the SAW using a non-standard da Vinci API interface.
- Stereo endoscopic video output is connected to the SAW.
- Two sets of stereo video outputs of the SAW are connected, one to each of the da Vinci master consoles.
- The supervisory surgeon has control of the patient-side manipulators and the system operates normally with respect to the his/her console interface.

**Trigger:** The supervisory surgeon depresses the master clutch pedal on the da Vinci surgical console and closes and then releases both MTM grips.

- Basic Flow:**
1. The surgeon closes and then releases both MTM grips in order to enter *Masters-as-Mice mode*.
  2. GUI mode becomes active and visible on both surgical consoles, a 3D pointer/cursor and menu system are overlaid onto the surgical console. Graphical tool icons appear at each of the PSM tools.
  3. The supervisory surgeon moves the 3D pointer within his/her field of view by manipulating one of the MTMs (the primary).
  4. The supervisory surgeon selects “mentor mode” by closing the primary MTM grip while the pointer is appropriately positioned on the graphical menu system.
  5. The menu system disappears from the trainee console and PSM control is transferred to the trainee. A telestration menu appears on the supervisory console.
  6. The trainee console behaves like a standard da Vinci system.
  7. The supervisory surgeon is able to telestrate by using his/her primary MTM as an input device.
  8. The camera clutch on the supervisory surgeon has shared control of the ECM. If both camera clutches are activated, then the trainee console takes precedence in order to direct the camera view.
  9. Menu options on the supervisory console allow the supervising surgeon to re-gain control of the PSMs, request control of the fourth arm, to control telestration and to overlay pre-operative image volumes.

**Post-conditions:** • Shared control of the PSMs.

### 3.5 Haptic Guidance: Virtual Fixtures

**Brief Description:** An interaction mode in which the surgeon shares control of the robot with the computer process. The goal of these task-dependent computer processes is to provide assistance to the surgeon by limiting the robot's motion within restricted regions and/or by influencing it to move along desired paths. The literature on virtual fixtures (VFs) classifies these behaviors as either forbidden region virtual fixtures (FRVFs) or guidance virtual fixtures (GVFs). FRVFs allow desired motion only in a predefined task space, whereas GVFs provide assistance in keeping the motion on desired paths or surfaces. In this architecture, FRVFs are defined using a fixed number of virtual planes, whereas GVFs can be selected from a predefined set of primitives.

**Pre-Conditions:**

- Endoscopic video outputs from the da Vinci system are connected to the SAW.
- Video output of the SAW is connected to the da Vinci master console.
- SAW is connected to the da Vinci system via Ethernet, using a non-standard API interface.
- The surgeon operates the master console. While the master clutch pedal is not pressed, the stereo display shows the live endoscopic images and operates normally.

**Trigger:** The surgeon depresses the master clutch pedal on the da Vinci surgical console.

**Basic Flow:**

1. The surgeon closes and then releases both MTM grips in order to enter Masters-as-Mice mode.
2. GUI mode becomes active and visible on the surgical console; a 3D pointer/cursor and menu system are overlaid onto the surgical console. Graphical tool icons appear at each of the PSM tools.
3. The surgeon moves the 3D pointer within his/her field of view by manipulating one of the MTMs (the primary).
4. The surgeon selects "virtual fixture mode" by closing the primary MTM grip while the pointer is appropriately positioned on the graphical menu system.
5. A new menu system appears on the surgeon console that allows adjustment of planes that define the boundary of forbidden regions. In this menu system, a fixed number of 3D planes are visible to the surgeon.
6. The surgeon can grab each of these planes as in MaM mode and

move the primary MTM to relocate these planes (See also medical image overlay, similar to adjustment of view plane).

7. The surgeon selects “done” button by closing the primary MTM grip while the pointer is appropriately positioned on the graphical menu system.
8. The surgeon releases the master clutch.

**Alternate Flow:** Instead of Steps 5 and 6, the pull-down menu opens to display a list of predefined virtual fixtures, listed by their unique identifiers. This feature is not currently supported by the daVinci API, but could be available with other robot systems.

**Post-conditions:** If MaM mode was not activated, the system operates normally (no GUI appears). If MaM mode was activated, the system operates as in the case of returning from clutch mode. The virtual fixture is now active, and motion of the robot is modified based on the properties of the selected virtual fixture.

### 3.6 Research Hardware: Snake Robot

**Brief Description:** The daVinci master console is used to control a snake robot system being developed for laryngeal surgery.

**Pre-conditions:**

- daVinci master console connected to the SAW.
- Snake robot connected to SAW, using one of the following options:
  - Snake robot control computer, running low-level control task, with network connection to the SAW.
  - Snake robot hardware (I/O boards) connected directly to the SAW, with low-level control task executing on the SAW.
- Stereo endoscopic video output connected to the SAW.

**Trigger:** Researcher starts application software that specifies use of daVinci master console and snake robot.

**Basic Flow:** This use case has the same basic flow as a standard daVinci system; the primary difference is that a different slave robot is controlled.

**Post-conditions:** None.

## 4 Logical View

### 4.1 Overview

A logical view of the subsystem architecture is presented in Figure 3. Its components are divided into five categories, namely physical devices, interfaces, real-time processes, application

logic and visualization. In this view, the relationship between components is indicated by the directed flow of command, state and event information, as an extension and generalization of the concept overview shown in Figure 1. While many of the blocks shown in Figure 3 will be implemented as distinct software modules, this view is primarily concerned with the arrangement and interaction between logic components. Therefore, specific implementation details are not described in detail until Section 7.

## 4.2 Communication Primitives

Figure 3 uses the following communication primitives between objects. These primitives are implemented using the Command Pattern, so that their behavior can vary depending on the types of objects involved and on the communication medium (e.g., shared memory or network). Detailed illustrations of these primitives are presented in Figure 4 (showing high-level task to low-level (device) task interaction), and Figure 5 (showing high-level task to device wrapper interaction). Note that although these figures show the interface with Device Drivers, the design of device drivers is outside the scope of this architecture. The architecture assumes that Device Drivers provide read/write/control methods and that they may also generate events that would be processed by a callback mechanism. It is noted that some operating systems (such as Linux) provide *condition variables* that are polled using the *select* system call; these can be wrapped by the CISST operating system abstraction library to generate events.

### 4.2.1 Command (cmd)

Commands are messages that are sent from one object to another. If the recipient is a Task, the message is queued in the task's Mailbox. If the recipient is a Device Wrapper, the corresponding "set" method is executed in the calling object's thread. There is no response to a command, nor is there any acknowledgment that it has been received or processed, although some limited error checking (such as mailbox full) could be done. If a response or acknowledgment is desired, it can be provided by an event.

### 4.2.2 State Read (state)

A state read occurs when a client object wishes to access data (state) from a server object. If the server object is a Task, this corresponds to a read from the State Data Table. Note that if the server object is on another machine, the data transfer requires serialization. If the server object is a Device Wrapper, a state read corresponds to a direct read from the device hardware or underlying device driver.

### 4.2.3 Event

An event enables one object (typically a "lower level" object) to inform another object (typically one or more "higher level" objects) about an "exceptional" condition that is usually

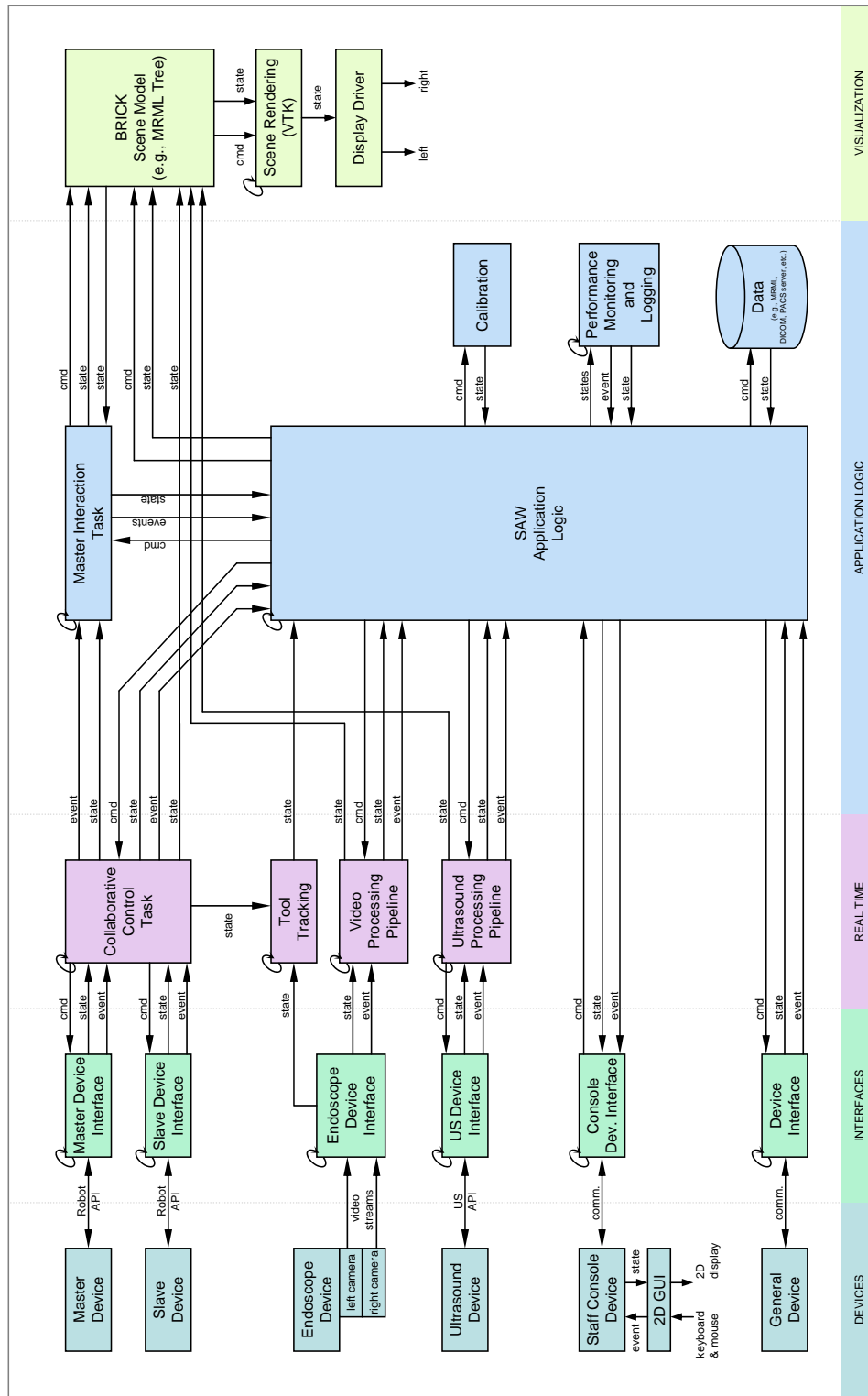


Figure 3: Sub-system architecture (Logical view).

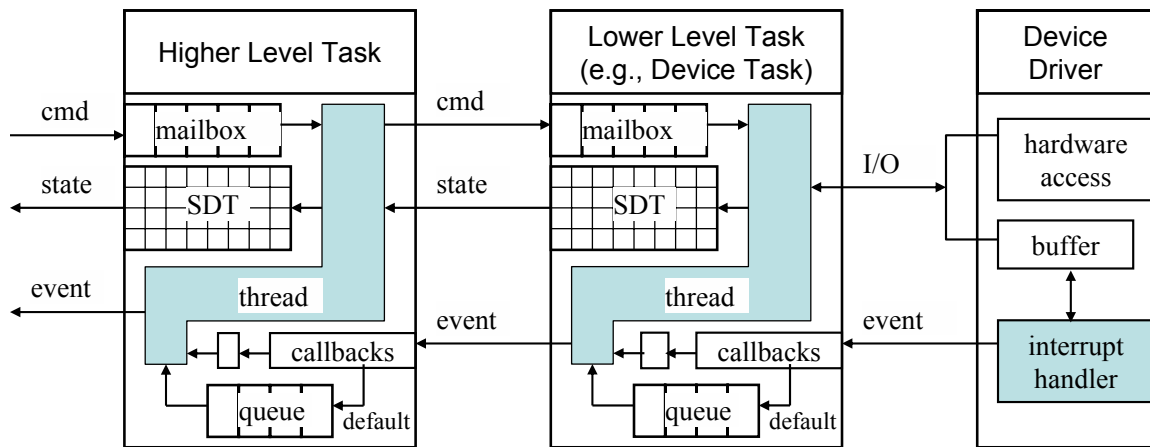


Figure 4: Interface between high-level and low-level (device) tasks.

detected asynchronously. Events are also implemented using the Command mechanism, where an Event Object is passed to a callback function that is specified by the receiving object. Because a Device Wrapper does not have its own thread, events are either propagated directly or invoked during a polling process. In the first case, a Device Driver event would invoke a Callback defined by the Device Wrapper, which would then invoke a Callback defined by the higher-level object, as shown in Figure 5. In this latter case, any call to a Device Wrapper method (e.g., any “get” or “set” method) could call the polling function, thereby causing the event to be propagated to the higher-level task.

### 4.3 Logical Components

#### Robotic Manipulators:

##### *Master Tool Manipulators:*

The master manipulators of the surgical console constitute the primary means of input and control for the surgeon. The “Master Tool Manipulators” (MTMs) of the ISI da Vinci system currently operate in three modes:

- Following mode: When in *following mode*, the patient-side slave manipulators follow the motion of the master manipulators and are teleoperated. The third *Patient Side Manipulator* (PSM-3) can be activated by tapping the clutch pedal. This allows the surgeon to toggle between PSM-3 and either PSM-1 or PSM-2, depending on which side PSM-3 is positioned.
- Master clutch mode: When the master clutch pedal is depressed, the system is taken out of *following mode*, such that PSM motion is no longer coupled to MTM motion.

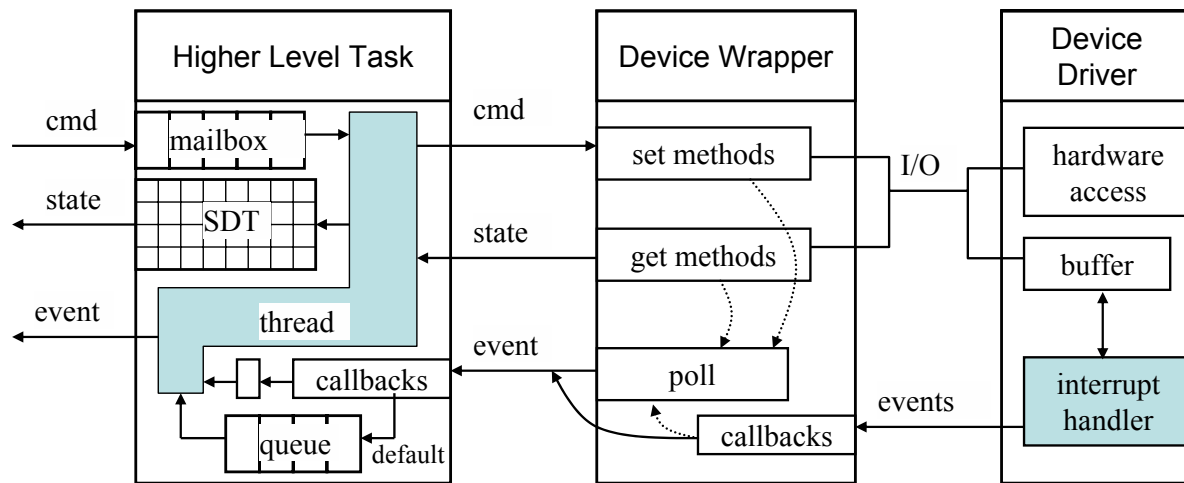


Figure 5: Interface between high-level task and device wrapper

During surgery, this allows the operator to re-center the MTMs within their range of motion, and thus increase the surgical workspace.

- Camera control mode: When the camera clutch pedal is depressed, the PSMs are taken out of *following mode* and control is transferred to the *Endoscopic Control Manipulator (ECM)* for camera repositioning.

The SAW framework adds a fourth mode—overlapped with master clutch mode—to allow the surgeon to interact with the SAW GUI. In this mode, each MTM operates as a 3D mouse, such that it can be used to position a graphical cursor overlaid on the stereo display console, while gripper open/close motions are used to emulate click and drag operations. In this way, the surgeon is able to interact with graphical objects and menus displayed by the SAW application. This mode is called *Masters as Mice (MaM)*.

#### *Patient Side Manipulators:*

When in *following mode*, the slave manipulators are teleoperated, based on synchronous input from the MTMs. Depending upon the teleoperator control system, force and/or motion signals may be reflected back to the master side from the slave manipulators. Within the SAW framework, both the master and slave manipulators are interfaced through a device interface, by means of a standardized robot API class.

#### **Collaborative Control:**

The collaborative control block couples the master and slave manipulators. In a single-slave, single-master configuration, this block implements teleoperation control. In general, an application may include multiple masters and/or slaves; therefore, the collaborative control

block provides a means to coordinate multiple manipulators. It contains a synchronous real-time loop for implementing control systems.

### **Video Processing:**

The workstation will support at least two types of video sources, namely:

- stereo endoscopy video, and
- ultrasound.

Support for video acquisition and processing pipelines will be provided.

### **Tool Tracking:**

A specialized video processing pipeline is used to implement visual tool/instrument tracking. As shown in Figure 3, this block receives state information from the collaborative control block in order to incorporate kinematic information into the tool tracking algorithm.

### **Brick Manager:**

The Brick Manager is the three dimensional analog of a standard window manager, in that it supports 3D user input and interaction with 3D graphical objects, such as image volumes and models, markers, annotations and in-situ video streams. The visual scene that is maintained by the Brick Manager is ultimately rendered in stereo for overlay onto the surgical console display. It can be used to provide intraoperative visualization and graphical user interface.

### **Master Interaction:**

The master interaction block facilitates user interaction with menu widgets and graphical scene objects represented by the Brick Manager. It provides the interface logic between the Master Manipulators and the Brick Manager when in *Masters-as-Mice mode*. Typical 2D windowing systems use the mouse input to create events (e.g., motion, click, release events) and bind callbacks to these events. The Master Interaction block provides a similar mechanism for the 3D MTM inputs by querying the state of the manipulators and listening for clutch events. The interaction logic transforms these inputs into pointer motion, button click events and specific behaviors such as object selection, dragging, rotation, resizing, etc. Possible pointer interaction logic elements for move and grab event handling are illustrated in Figures 6, 7, and 8. We note, however, that the architecture does not impose an event-based programming model. For example, the rendering task can update the positions of the 3D pointers by using the **State Read** mechanism (Section 4.2.2) to query the positions of the MTMs, rather than by relying on events generated by the Master Interaction task. When using the da Vinci surgical console, the master manipulators will

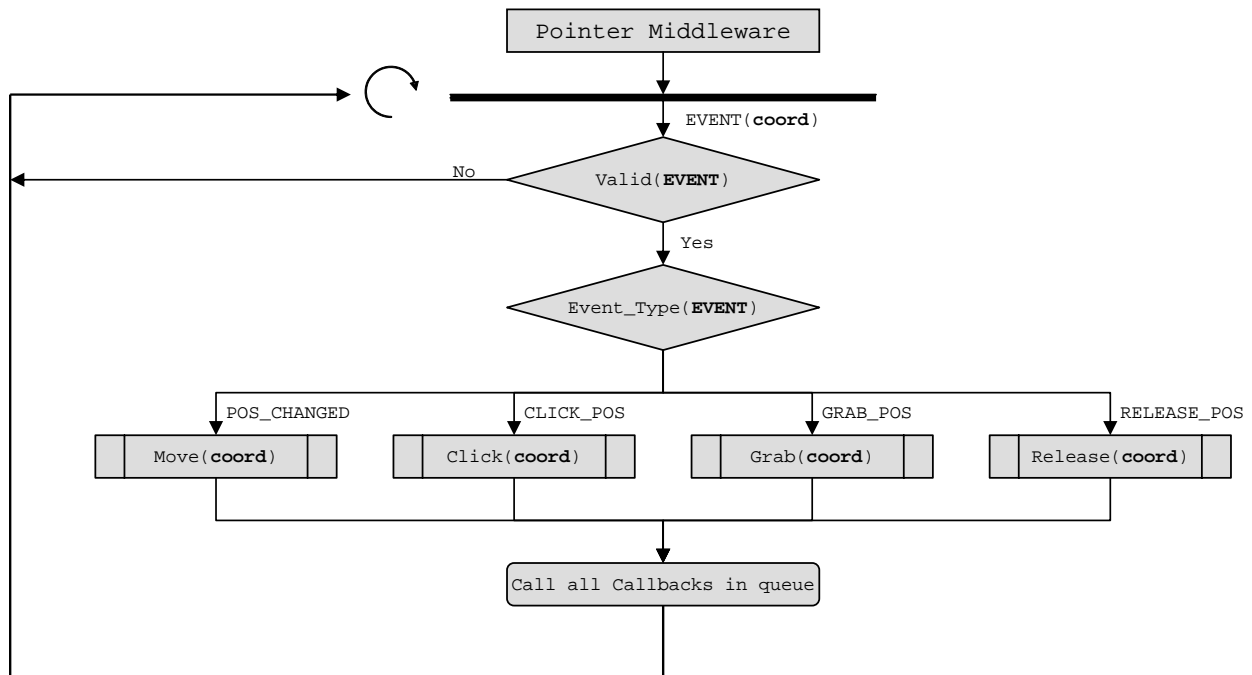


Figure 6: Basic 3D pointer interaction logic—Event Handling

be used as input devices for user interface within the surgical console. We will initially overlap this mode with the existing master clutch mode of the da Vinci in the following way:

---

PROCESS\_MTM\_EVENT(*Event*)

```

1  if Event == MASTER_CLUTCH_PRESSED
2      then InitMTMPos = GETMTMPOS()
3           WAIT(3 seconds)
4           ClutchState = GETMASTERCLUTCHSTATE()
5           MTMPos = GETMTMPOS()
6           if (ClutchState == PRESSED) and ((MTMPos - InitMTMPos) < ε)
7               then ENTERSAWCONSOLEMODE()
8               else return
9  else return
    
```

---

While in *Saw Console Mode* the position of the MTM will be used to drive the 3D pointer, while its gripper handle will be used as a button.

### Application Logic:

Application-specific logic is encapsulated in this block and is defined by the application developer within the scope of the SAW application framework. Once the “Master Interaction”

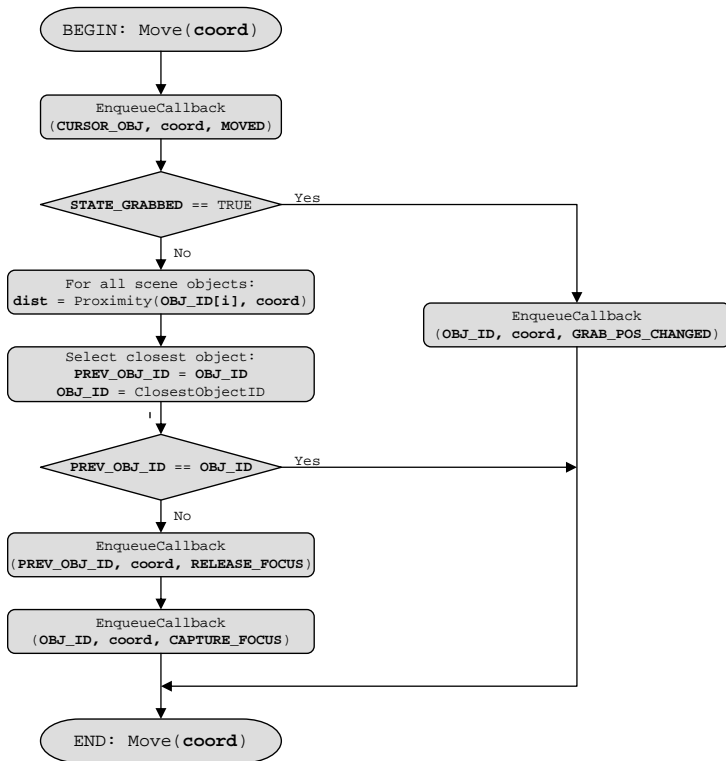


Figure 7: Basic 3D pointer interaction logic - Move Event

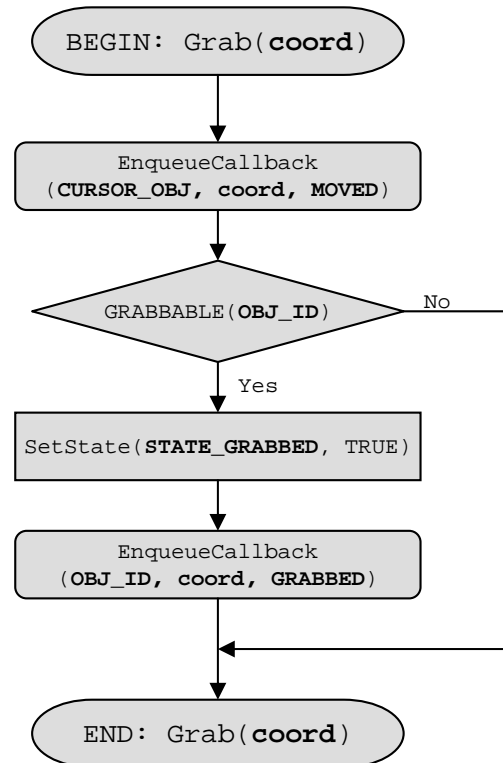


Figure 8: Basic 3D pointer interaction logic - Grab Event

component has determined which “widget” is currently active, all events will be forwarded to the “widget” and its logical layer. If the application requires a more direct access to the MTMs, the application will be able to access the MTM’s state and disable the event forwarding from the “Master Interaction” component.

### Calibration and registration:

The application framework will provide standard tools and algorithms for device calibration and coordinate registration. Examples of typical calibration and registration tasks include:

- Kinematic calibration of robot manipulators,
- Calibration of navigation systems,
- Ultrasound calibration,
- Model-to-video registration.

### Performance Monitoring and State Logging:

This component can be used to observe system states and performance metrics for safety and logging purposes. A state recovery mechanism will provide the means to restore specific system states.

### Staff Console:

The staff console can be used to interact with the application from outside of the surgical console. It can be used to perform tasks that are not accessible within the Brick Manager, such as data preparation, as well as to monitor application states. It may also include the CISST Interactive Research Environment (IRE).

### Device Interfaces:

Robot manipulators, video sources, external consoles and other peripherals are categorized as devices, and as such are interfaced to the application framework by means of device interfaces. These device-specific blocks create a layer of abstraction between external hardware or software modules in order to present a uniform interface to the application logic.

## 5 Process View

### 5.1 Overview

Figure 9 depicts the concurrent units of execution in the system. In general, these execution units are provided by threads (e.g., multi-threading), rather than by multiple processes. Note, however, that the “low-level robot control” may be provided externally (e.g., when connected to an external device using the vendor’s API). In this case, it would be a separate process, possibly on a separate computer. Similarly, signal and image processing pipelines may be distributed as external processes on separate computing hardware.

### 5.2 Process Components

| Component                               | Description  |
|---|--|
| Low-level Robot Control:                | Manipulator-level robot control system. In the SAW architecture, this will typically be an external process that is supported on a dedicated computational platform.                 |
| High-level Collaborative Robot Control: | A control system process that synchronously couples master interfaces and slave manipulators.  |
| Surgical Console UI Interaction:        | An interactive intraoperative 3D graphical user interface. The GUI is intended to augment the master surgical interface for enhanced image visualization and control by the surgeon. |

|                                   |  |
|-----------------------------------|--|
| Scene Rendering:                  | A graphical rendering pipeline responsible for stereo visualization and overlay in the surgeon's console.  |
| Signal/Image Processing Pipeline: | A processing pipeline that is used for video processing—such as instrument tracking and image overlay—and other signal processing tasks. This pipeline may include the acquisition of images, video, and signals that originate from external devices or distributed system components. For some applications it may be necessary to perform computationally demanding or specialized processing on a dedicated hardware system; therefore, this component can also be an external signal processing system. Data flow and interfacing between such distributed components will be handled at the <i>interface handler level</i> . |
| Interface Handlers:               | This is an interface layer for communicating events and state data between processes within the application framework.   |
| Staff GUI:                        | Process for managing a staff user interface that is distinct from the surgeon's console. This includes an interactive shell (IRE) for accessing and modifying system states from a Python interpreter, for use primarily during application development.   |
| Monitoring and Logging:           | A process that monitors and logs system states, signals fault conditions and facilitates state recovery. It will act as a watchdog for all key system tasks, and as such, is a key safety mechanism in the SAW architecture.   |
| Application Logic:                | The core of the application framework.   |
| User Application:                 | This process describes the specific clinical application logic, as implemented by the developer.   |

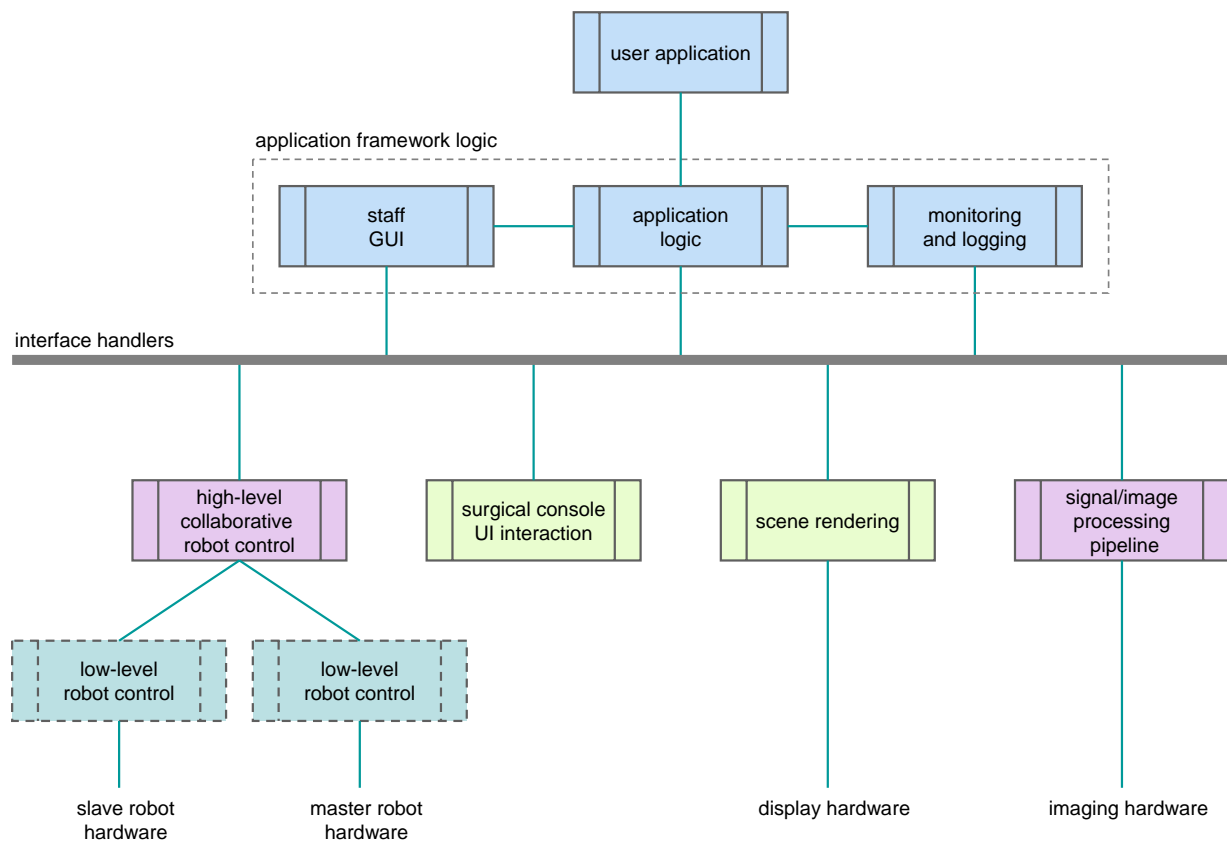


Figure 9: Sub-system architecture (Process View).

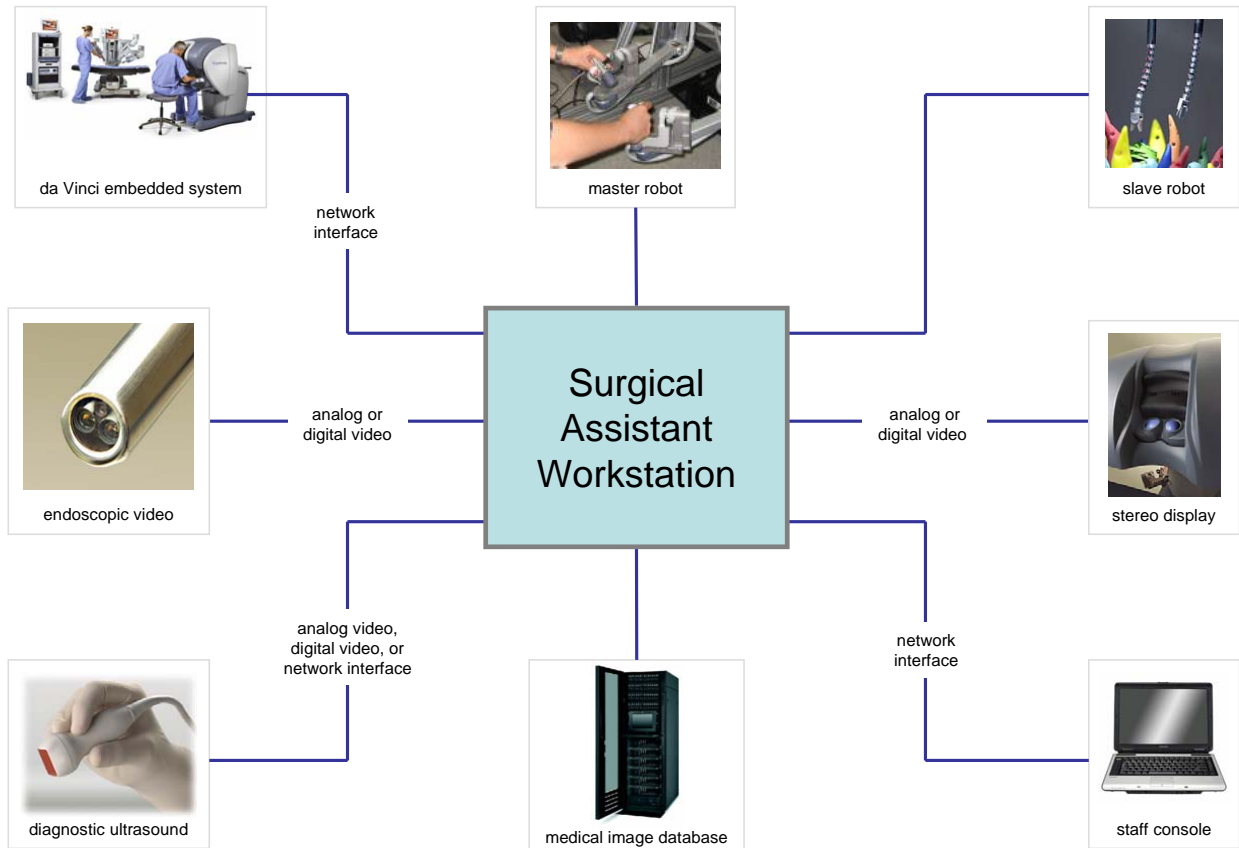


Figure 10: Sub-system architecture (Deployment View).

## 6 Deployment (Physical) View

### 6.1 Overview

Deployment of the SAW framework is application specific. Figure 10 shows a generic deployment view that illustrates a number of anticipated components and sub-systems.

### 6.2 Physical Components

| Sub-System                      | Hardware Interface | Description   |
|---------------------------------|--------------------|---|
| Surgical Assistant Workstation: | -                  | Workstation that runs the core SAW application logic. |

|                           |   |   |
|---------------------------|---|---|
| Da Vinci Embedded System: | network interface                                       | The embedded system contained within the da Vinci telesurgical robot. This system manages the Master Tool Manipulators, Patient-Side Manipulators and consoles. |
| Master Robot:             | various   | Placeholder for a research-grade master interface device. Examples of such devices include the CISST MTMs, and Steady Hand Robot.                               |
| Slave Robot:              | various   | Placeholder for a research-grade slave manipulator device. Examples of such devices include the CISST PSMs, Snake Robot, and RCM.                               |
| Endoscopic Video:         | analog/digital video                                    | Endoscopic stereo video system.   |
| Diagnostic Ultrasound:    | analog/digital video, or vendor-specific API interface. | Diagnostic ultrasound system.   |
| Stereo Display:           | analog/digital video                                    | Stereo display system, e.g., da Vinci console, head mounted displays, etc.  |
| Medical Image Database:   | file or network (e.g., PACS)                            | Source of medical images, models, surgical plans and other application data. For example, this could include a clinical PACS system.                            |
| Staff Console:            | network   | A remote console that functions as a user interface for operating room support staff.   |

## 7 Implementation (Development) View

### 7.1 Overview

Figure 11 depicts a hierarchical view of the core SAW software libraries and their dependencies. The bottom rows contain the CISST foundation libraries, as well as external packages such as Python, LAPACK, and the daVinci research API. The `cisstDeviceInterface` library includes the abstract base class for all device interfaces, whether Device Tasks or Device Wrappers. Specific device interfaces are derived from this class. Similarly, `cisstRobot` defines generic robot capabilities, whereas robot-specific implementations are provided by modules such as `cisstISI` (for the Intuitive Surgical daVinci robot). The figure also shows higher-level functionality such as video processing, instrument tracking, and collaborative robot control. All of this is encompassed by the SAW application framework.

### 7.2 Development Components

|                      |  |
|----------------------|--|
| cisstCommon          | Common tools such as logging, error handling, serialization, and class and object registries.  |
| cisstInteractive     | The Interactive Research Environment (IRE) allows C++ code/data to be accessed and modified from the Python interpreter.   |
| cisstVector          | Provides support for linear algebra using vectors and matrices (fixed size and dynamic), quaternions, and transformations in two and three dimensions.   |
| cisstOSAbstraction   | Operating system abstractions for threads, semaphores, mutex, timers, dynamic loading.   |
| cisstImage           | An image class—based on the cisstVector dynamic matrix—that implements image processing algorithms.  |
| cisstNumerical       | C++ interface to numerical computation libraries, such as LAPACK, MINPACK, Lawson-Hanson, etc.   |
| cisstDeviceInterface | Interfaces to specialized hardware, such as motion controllers, force sensors, I/O boards, etc.  |
| cisstISI             | A wrapper class that encapsulates ISI API functions with cisst-library-compatible interfaces and emulating these functions for non-daVinci hardware, where appropriate.  |
| cisstRealTime        | Real-time support, including tasks and state data tables.  |
| cisstStereoVision    | Algorithms for managing stereo image pairs and geometry.   |
| Python               | A high level programming language; see: <a href="http://www.python.org">www.python.org</a> .   |
| LAPACK               | The Linear Algebra PACKage is a library for numerical computing; see: <a href="http://www.netlib.org/lapack">www.netlib.org/lapack</a> .   |
| da Vinci API         | The da Vinci application programming interface provides read access to a selection of states in the da Vinci embedded system.  |
| OpenGL               | The Open Graphics Library is a standard specification defining a cross-language cross-platform API for writing applications that produce 3D computer graphics. See: <a href="http://www.opengl.org">www.opengl.org</a> . |
| VTK                  | The Visualization Toolkit. See: <a href="http://www.vtk.org">www.vtk.org</a> .   |
| Calibration          | Methods for kinematic calibration, camera/image calibration, ultrasound calibration, etc.  |
| Device Interfaces    | Prototype interface for external devices. Each device interface instance will implement a specific peripheral device or data source.   |
| cisstRobot           | Uniform interface to a number of research robots.  |
| Video Processing     | A video processing pipeline for handling input from endoscopic video, diagnostic ultrasound and other imaging modalities.  |
| Brick Manager        | A 3D scene manager for the surgeon console.  |

|                           |   |
|---------------------------|---|
| UI Interaction            | The core interaction logic that defines the operation of the user interface at the surgeon console. This component manages user input from the master interface and interprets this input with respect to scene objects managed by the Brick Manager. |
| Collaborative Control     | A teleoperation control system that couples master and slave mechanisms. In the SAW framework, this system is not limited to a single master-slave pair, but can support multiple manipulators in a collaborative way.                                |
| Instrument Tracking       | Patient-side instrument/tool tracking based on vision and kinematic information.  |
| SAW Application Framework | The overall application framework.  |

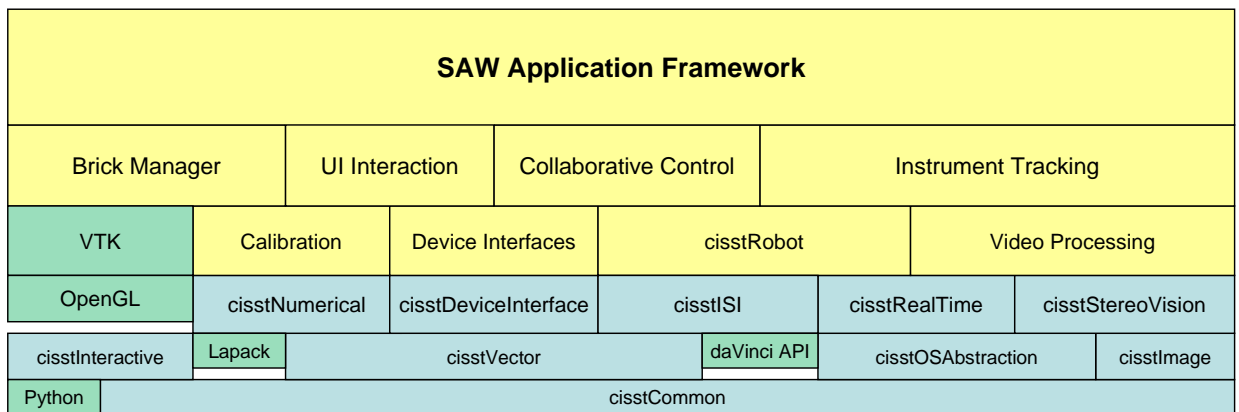


Figure 11: Sub-system architecture (Development View).