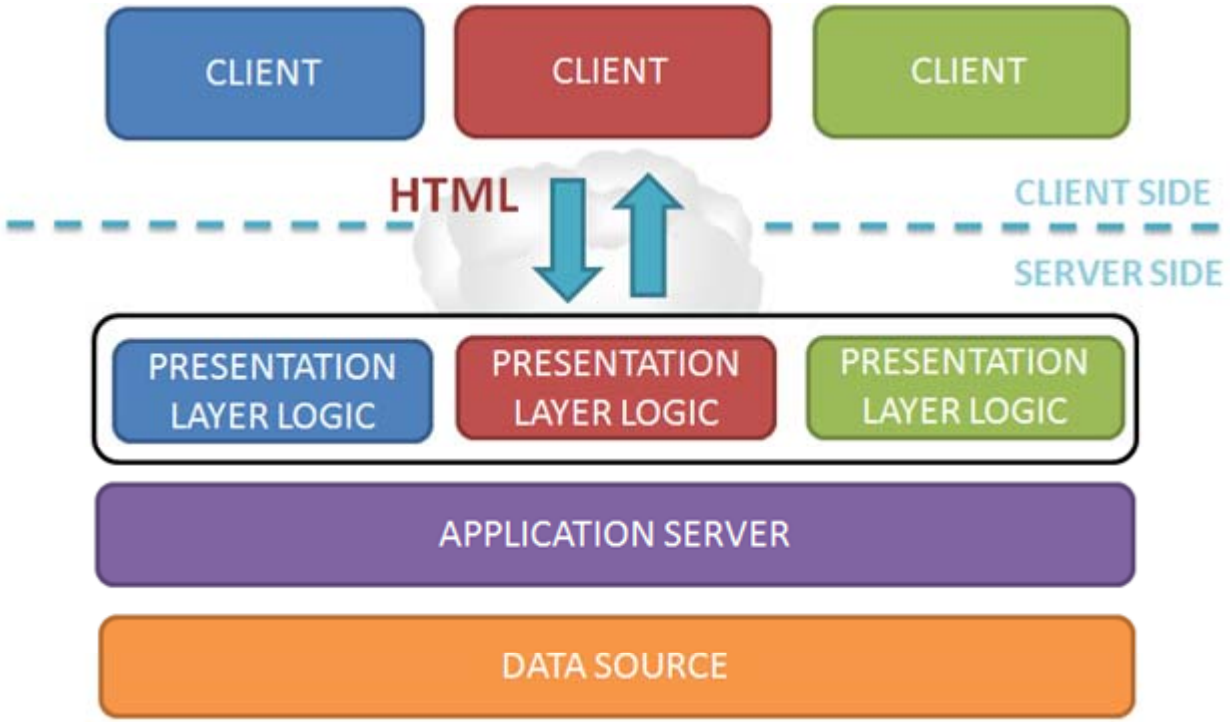
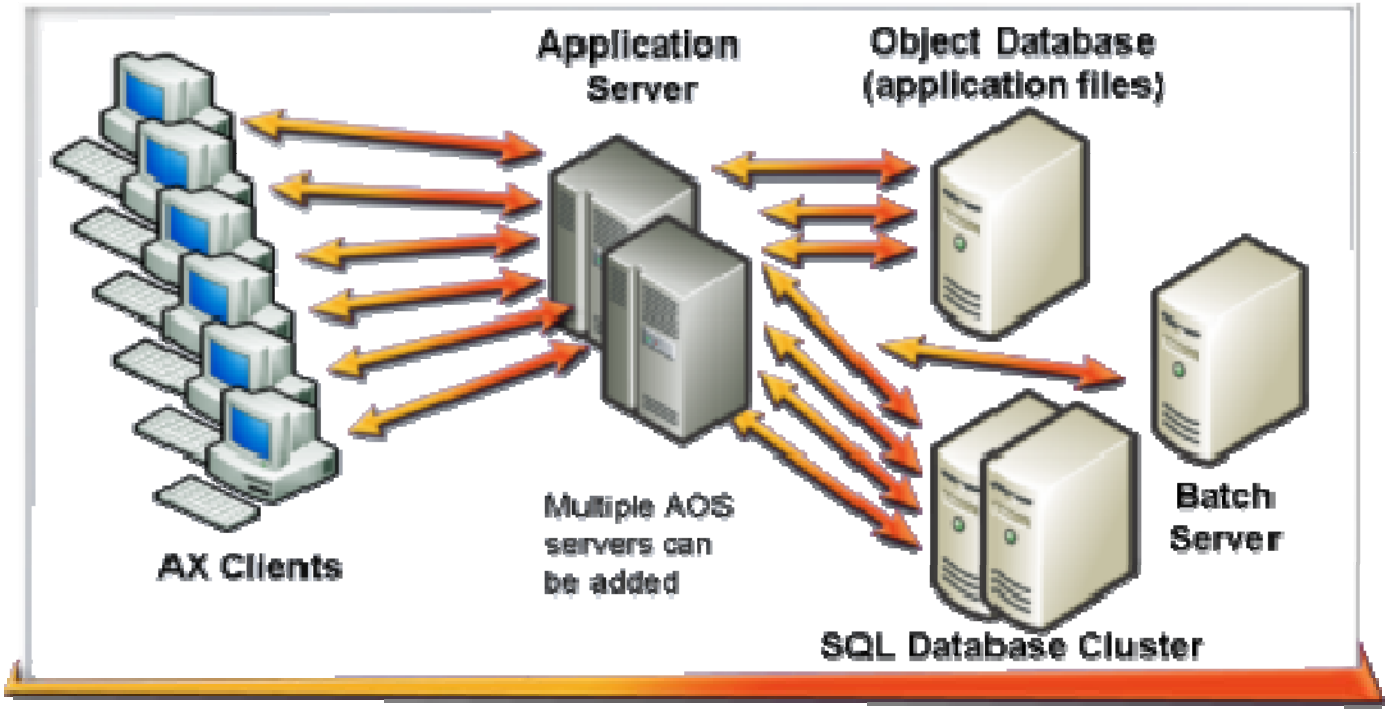
CPSC 875

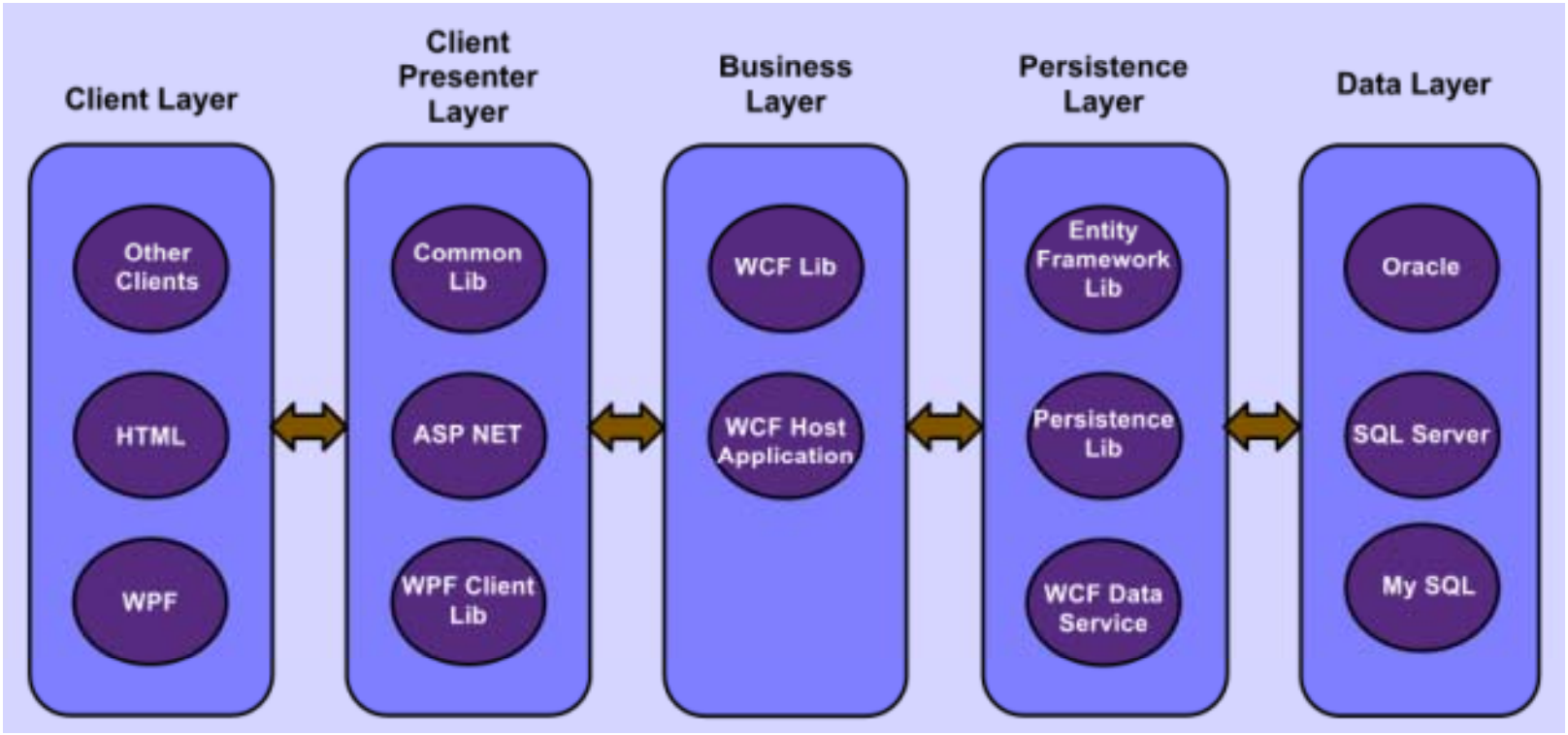
John D. McGregor

C 8 More Design

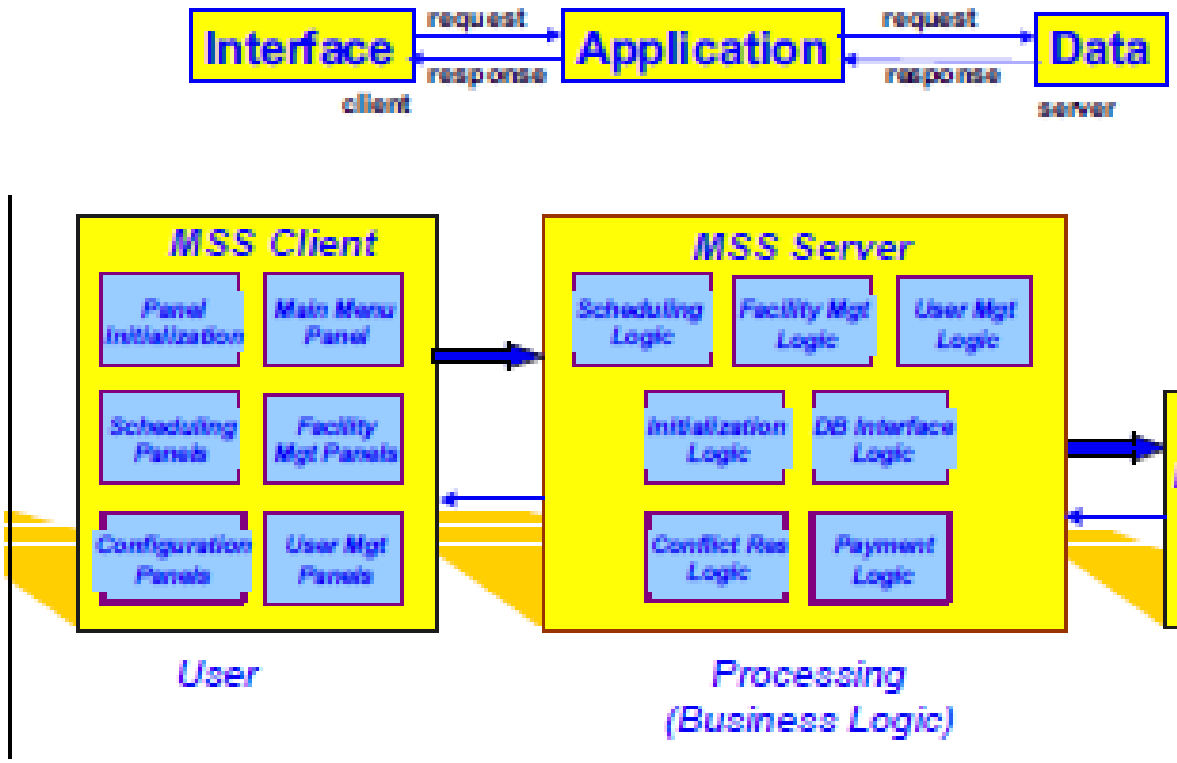


OLD SKOOL WEB APPLICATION ARCHITECTURE

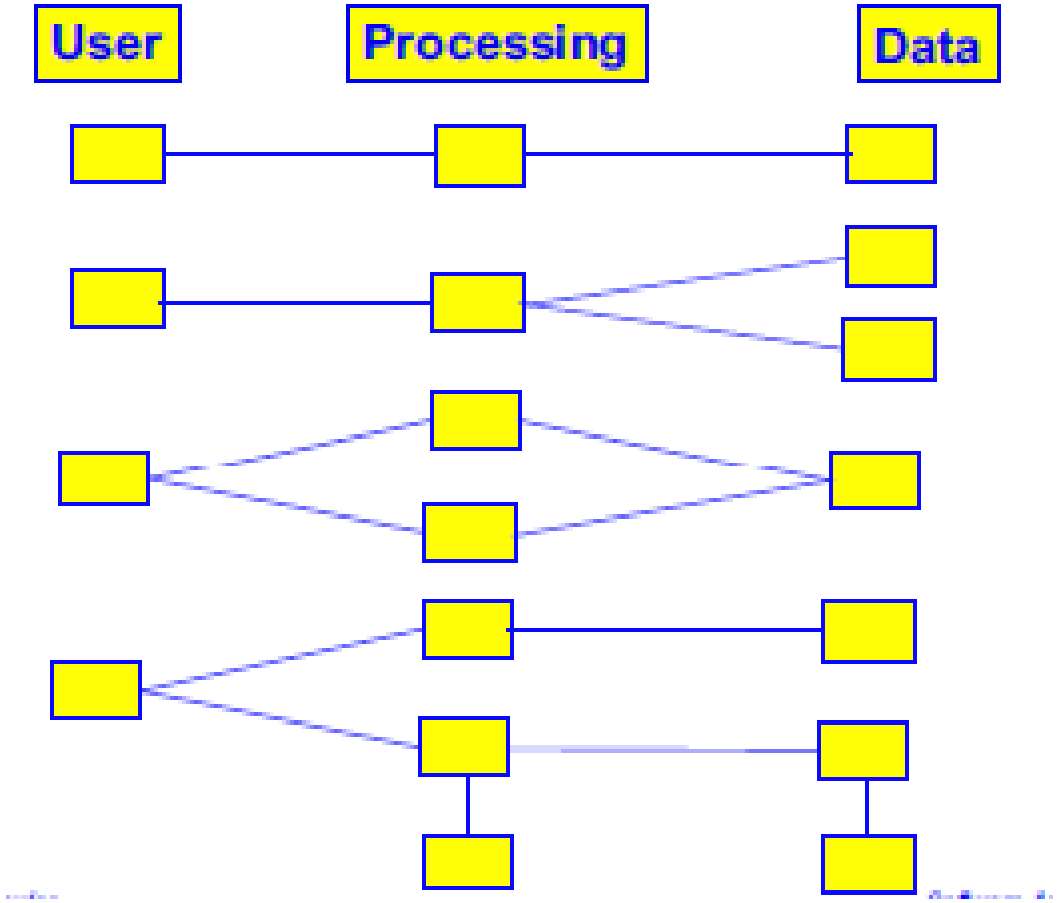

Application Server
Object Database (application files)
AX Clients
Multiple AOS servers can be added
Batch Server
SQL Database Cluster


Client Layer
Client Presenter Layer
Business Layer
Persistence Layer
Data Layer
Other Clients
HTML
WPF
Common Lib
ASP NET
WPF Client Lib
WCF Lib
WCF Host Application
Entity Framework Lib
Persistence Lib
WCF Data Service
Oracle
SQL Server
My SQL

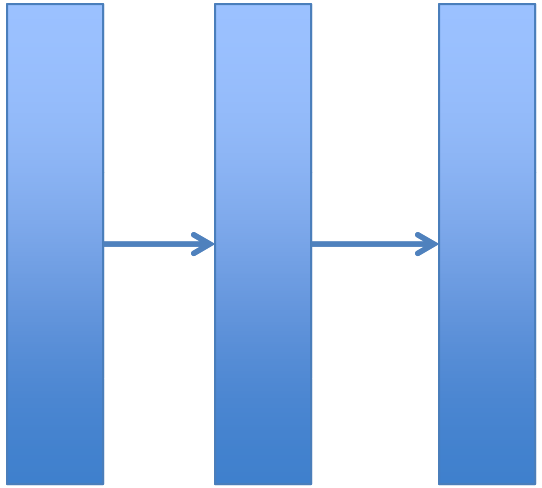
3-tier


Interface
request
response
client
Application
request
response
Data
server
MSS Client
Panel Initialization
Main Menu Panel
Scheduling Panels
Facility Mgt Panels
Configuration Panels
User Mgt Panels
MSS Server
Scheduling Logic
Facility Mgt Logic
User Mgt Logic
Initialization Logic
DB Interface Logic
Conflict Res Logic
Payment Logic
Database Server
User
Processing
(Business Logic)
Data

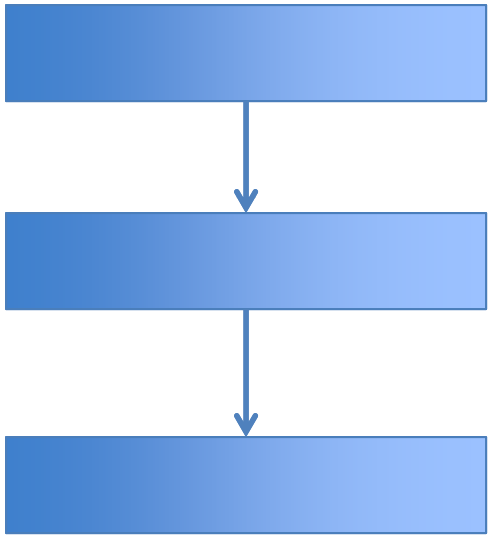
Variations



Tier vs Layer

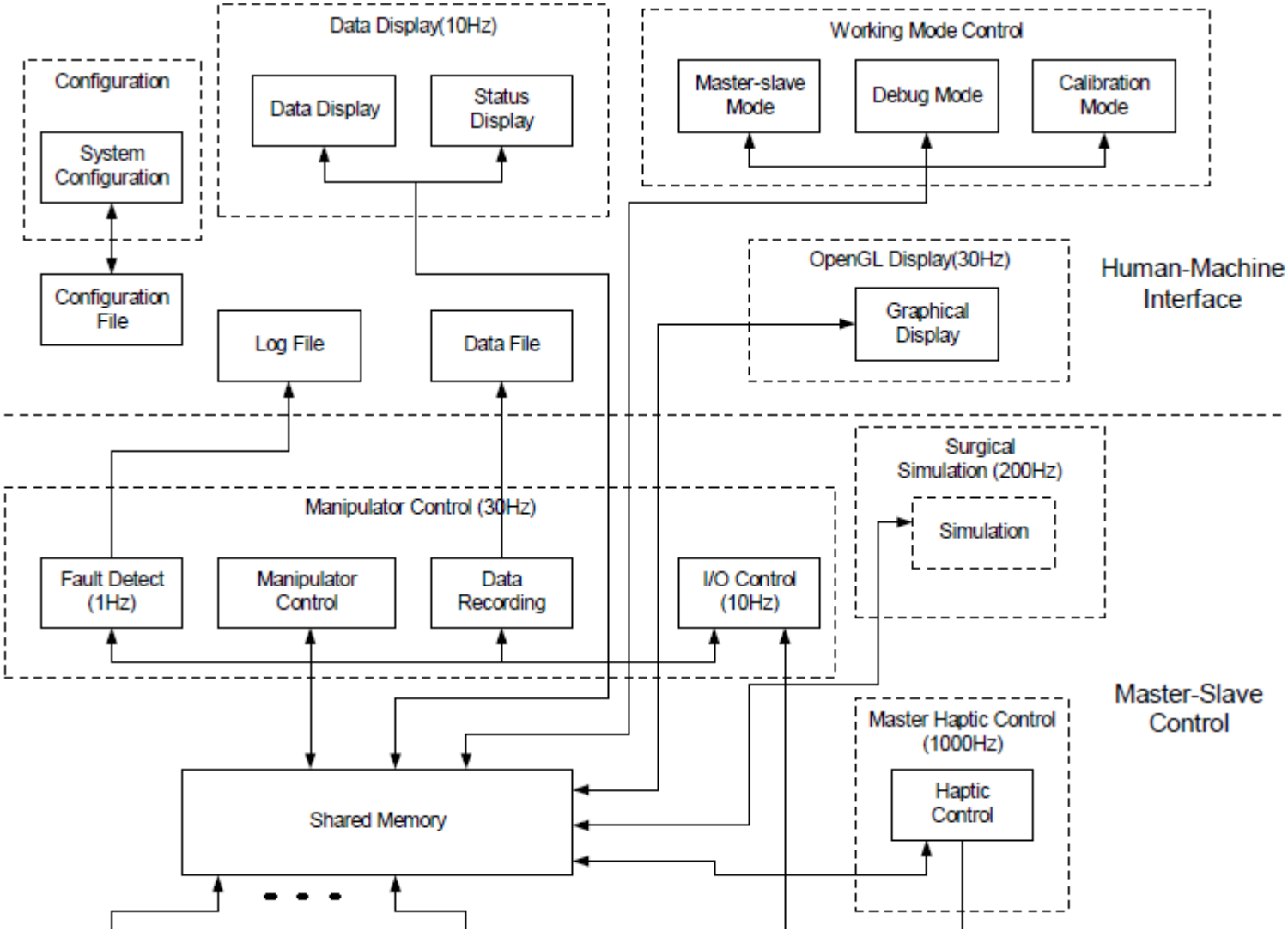


Abstracting away from user



Abstracting away from hardware

Modes


Data Display(10Hz)
Working Mode Control
Configuration
Data Display
Status Display
Master-slave Mode
Debug Mode
Calibration Mode
System Configuration
OpenGL Display(30Hz)
Human-Machine Interface
Graphical Display
Configuration File
Log File
Data File
Surgical Simulation (200Hz)
Simulation
Manipulator Control (30Hz)
Fault Detect (1Hz)
Manipulator Control
Data Recording
I/O Control (10Hz)
Master-Slave Control
Master Haptic Control (1000Hz)
Haptic Control
Shared Memory

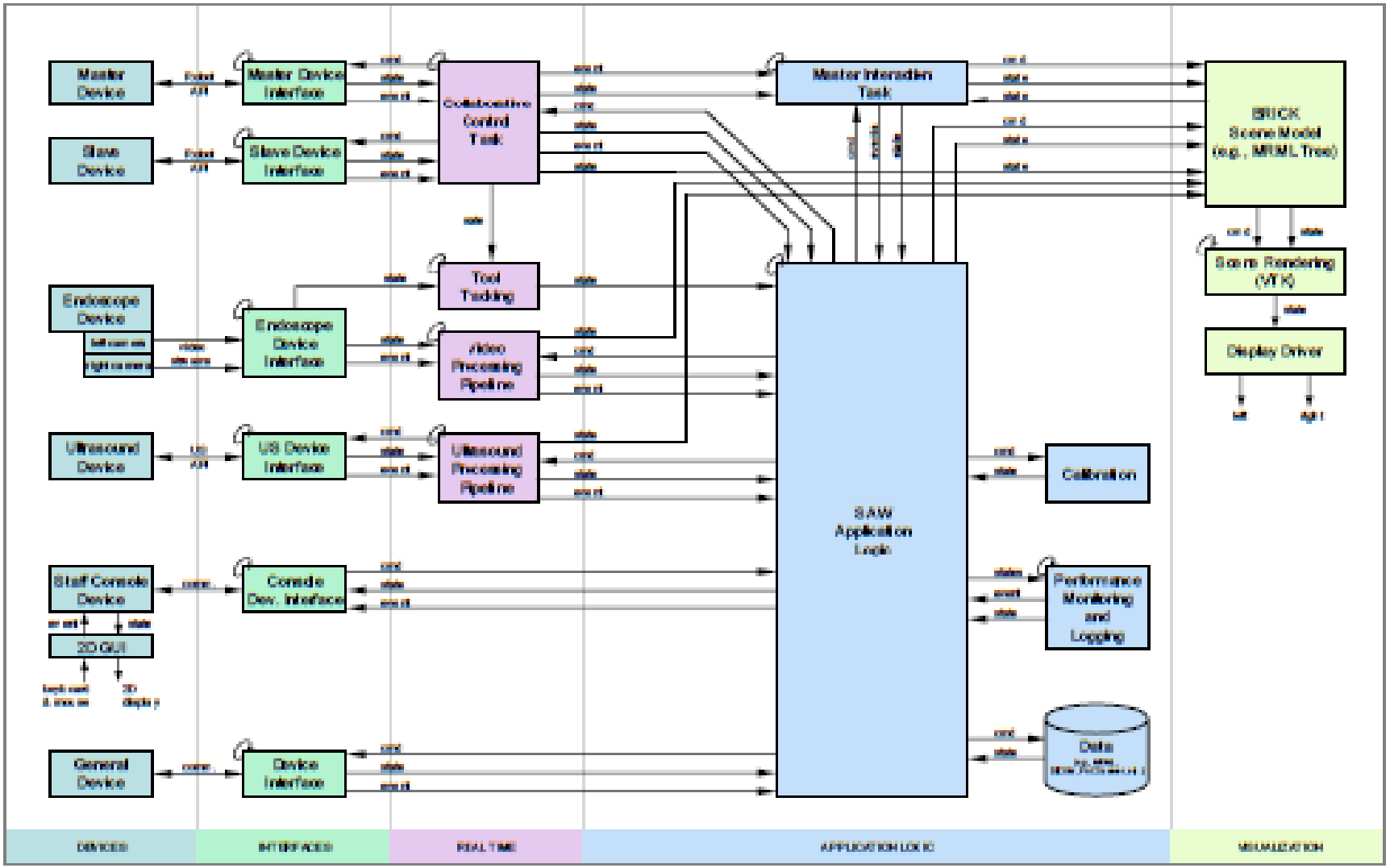
Modes

- Each mode can lead to a very different internal path
- The “state” pattern encapsulates the logic for a given state in a module and swaps out the state module with each change of mode
- The module that encapsulates the entire state machine is a higher level

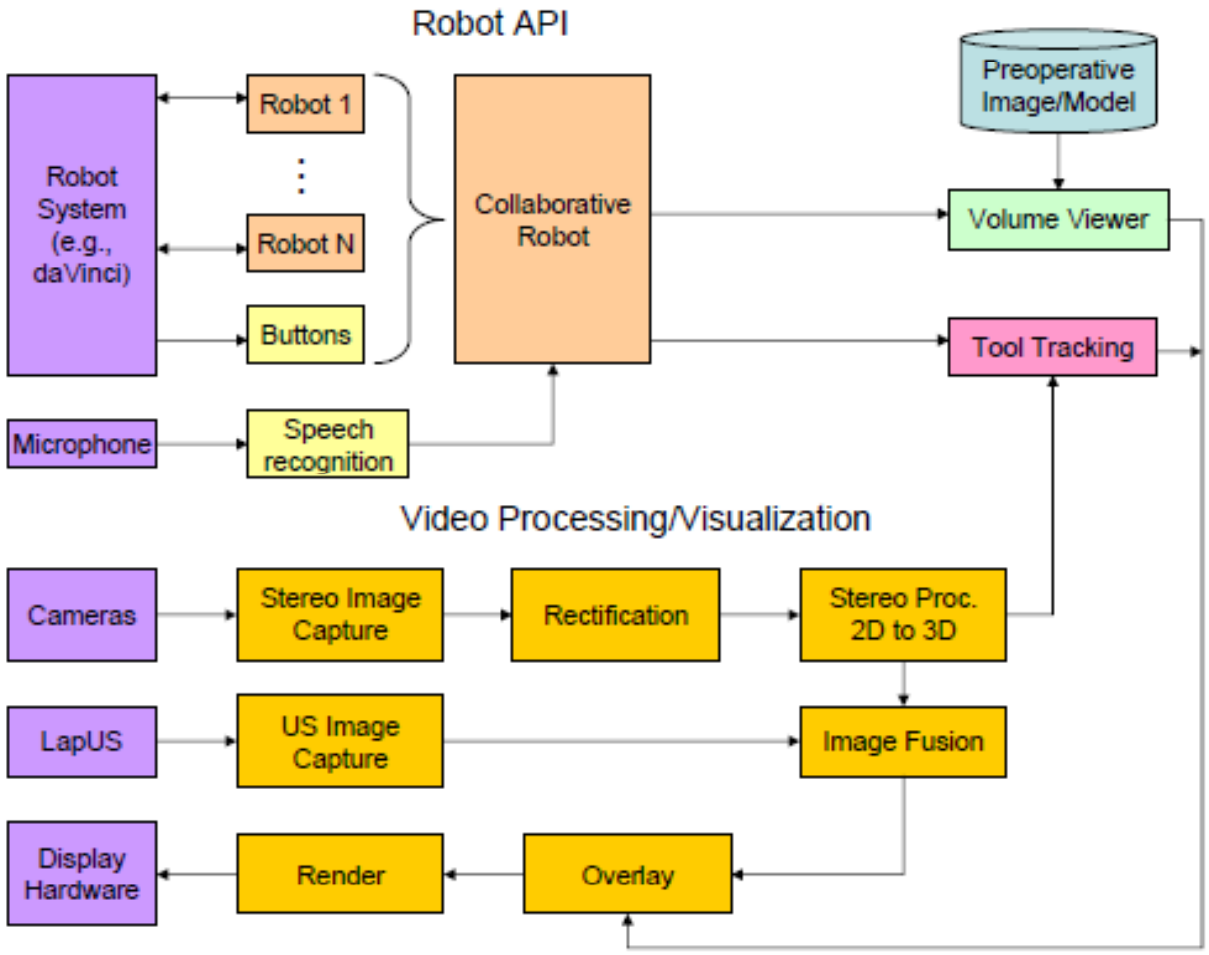
Modes

- What is the purpose of the mode?
- What capabilities need to be “live” in the mode?
  - For the Debug mode all instruments are turned off
- What is local data and what is global (to the state machine).

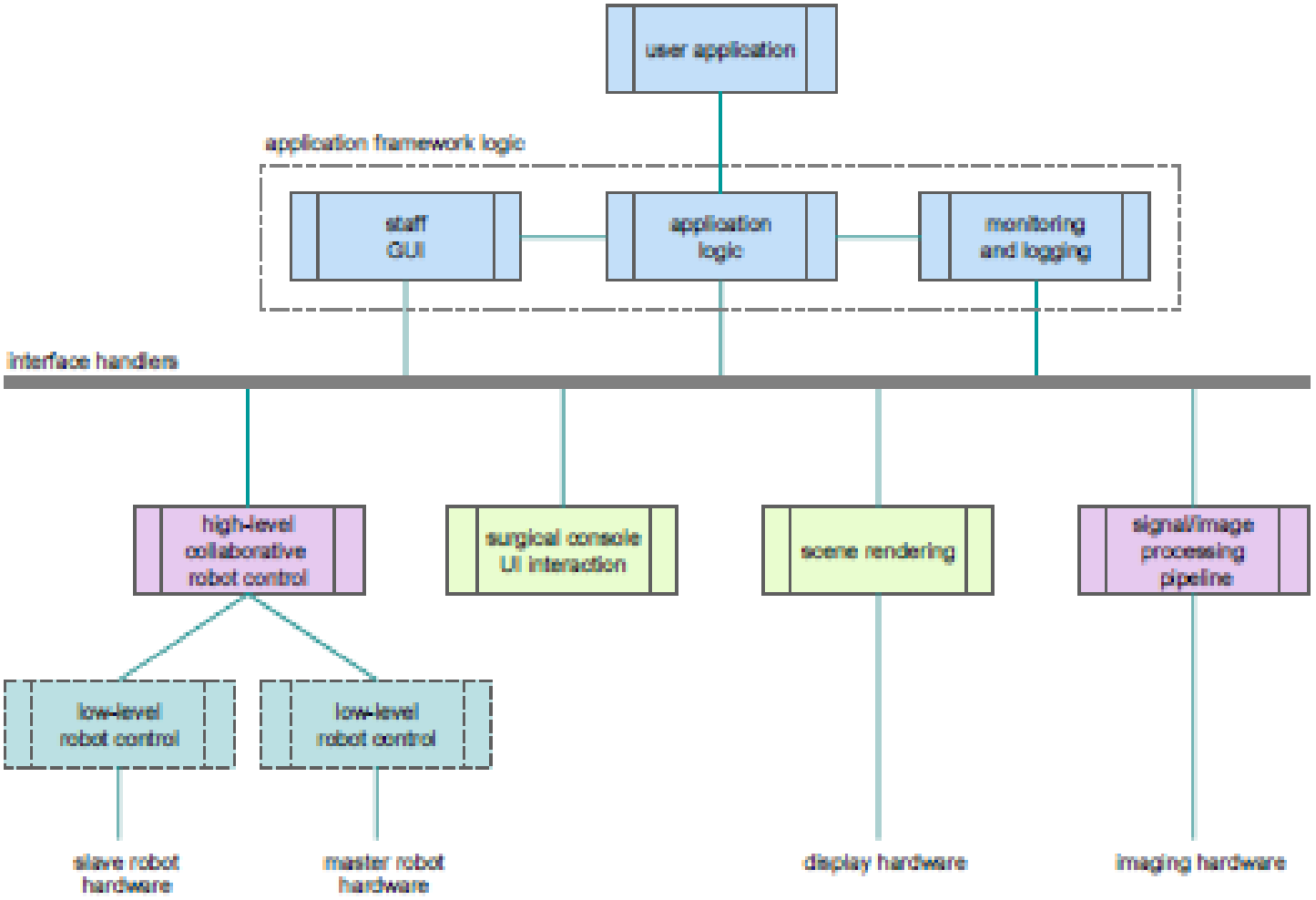
Use of symmetry

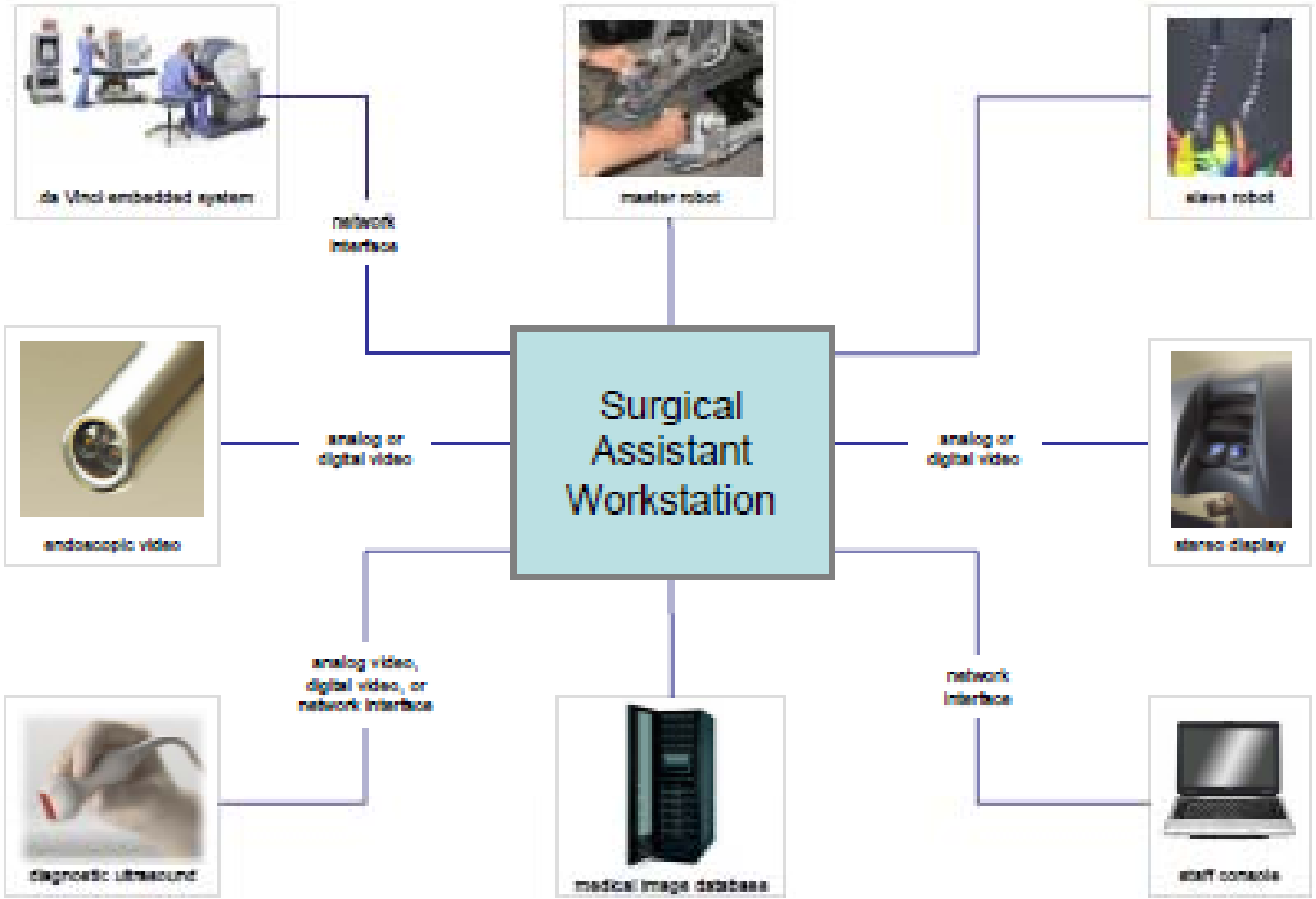


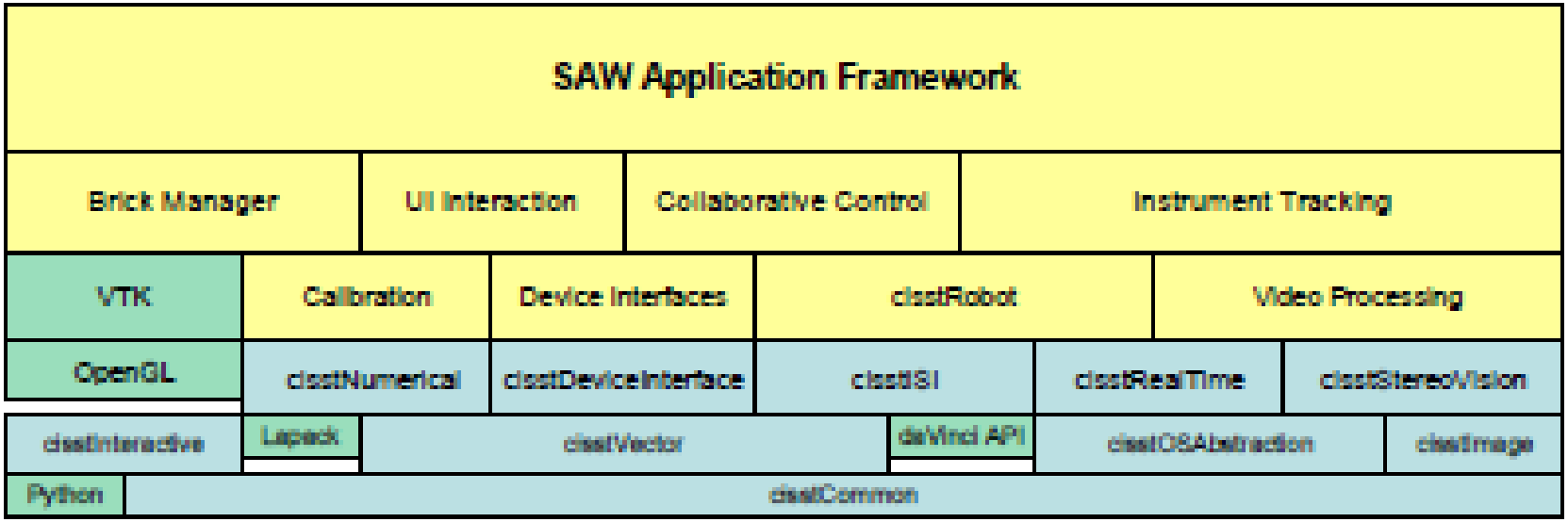
Use of color


Robot API
Robot System (e.g., daVinci)
Robot 1
Robot N
Buttons
Collaborative Robot
Preoperative Image/Model
Volume Viewer
Tool Tracking
Microphone
Speech recognition
Video Processing/Visualization
Cameras
Stereo Image Capture
Rectification
Stereo Proc. 2D to 3D
LapUS
US Image Capture
Image Fusion
Display Hardware
Render
Overlay

Use of legends




da Vinci embedded system
network interface
master robot
slave robot
endoscopic video
analog or digital video
Surgical Assistant Workstation
analog or digital video
stereo display
analog video, digital video, or network interface
network interface
diagnostic ultrasound
medical image database
staff console

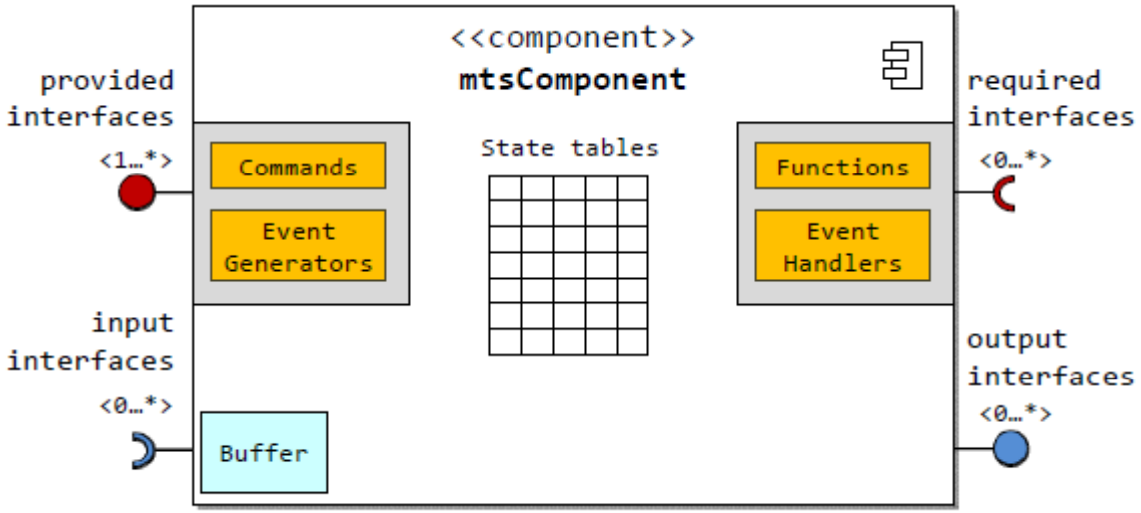

SAW Application Framework
Brick Manager
UI Interaction
Collaborative Control
Instrument Tracking
VTK
Calibration
Device Interfaces
cisstRobot
Video Processing
OpenGL
cisstNumerical
cisstDeviceInterface
cisstSI
cisstRealTime
cisstStereoVision
cisstInteractive
Lapack
cisstVector
daVinci API
cisstOSAbstraction
cisstImage
Python
cisstCommon

Timing

- For example, the control period of the haptic devices is 1ms to keep the smooth control response for the user, and the control period of the error detection is the slowest. The control rate for the manipulator motion control is 30Hz, which is limited by the maximum communication frequency (about 35Hz at 19200bps RS-232 baud rate) between the computer and the motor driven units. The control frequencies of the different software modules are follows:
  - Manipulator motion control: 30Hz
  - Error detection module: 1Hz
  - I/O control module: 10Hz
  - Haptic device control: 1000Hz
  - Surgical simulation: 200Hz

Component model

- Physical, component model



Component model

- A cisst component contains lists of provided interfaces, required interfaces, outputs, and inputs, as shown in Figure 1.
- Each provided interface can have multiple command objects which encapsulate the available services, as well as event generators that broadcast events with or without payloads.
- Each required interface has multiple function objects that are bound to command objects to use the services that the connected component provides. It may also have event handlers to respond to events generated by the connected component.
- When two interfaces are connected to each other, all function objects in the required interface are bound to the corresponding command objects in the provided interface, and event handlers in the required interface become observers of the events generated by the provided interface.
- The output and input interfaces provide real-time data streams; typically, these are used for image data (e.g., video, ultrasound).

Local/Global Component Manager

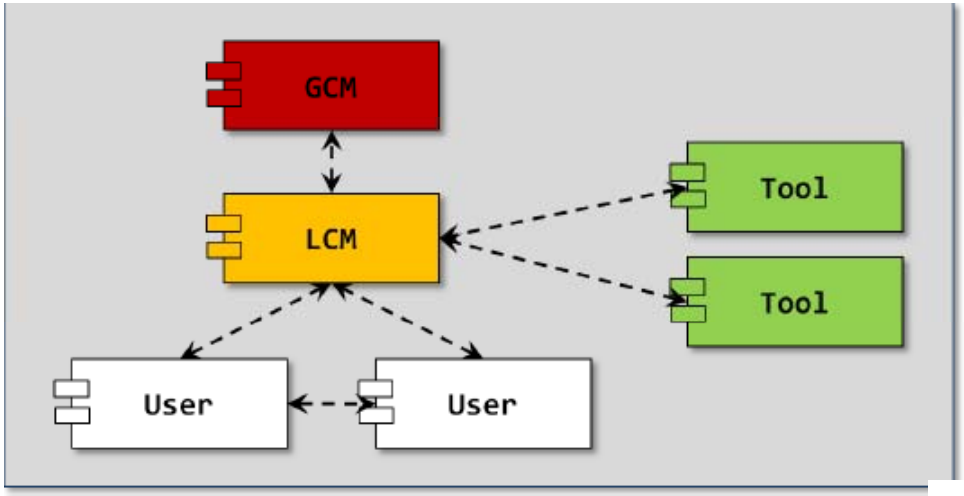
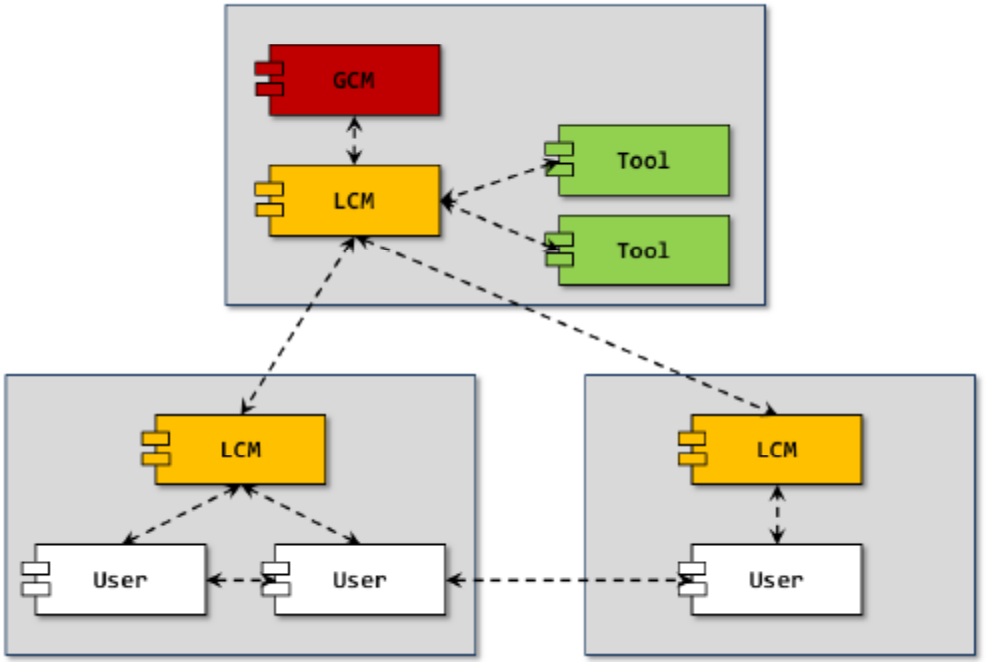
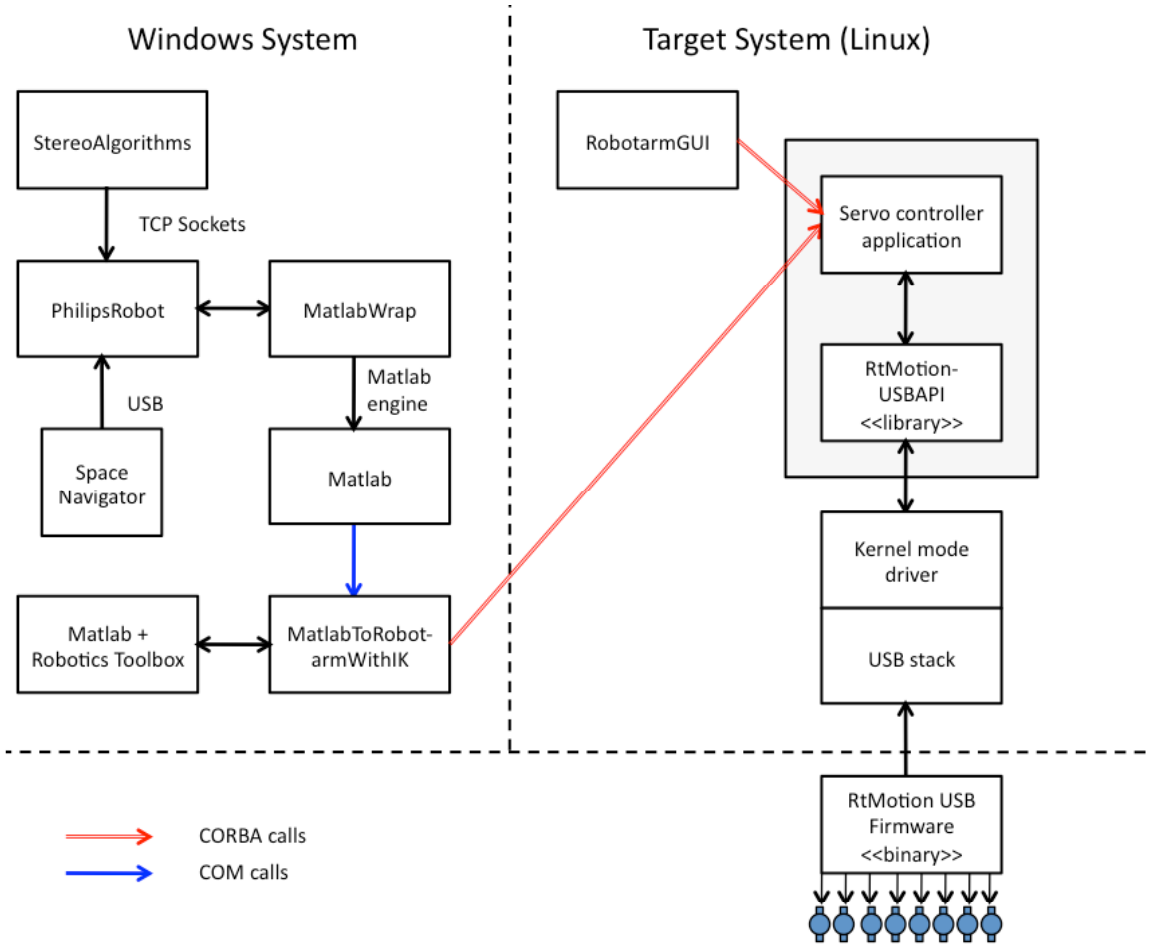
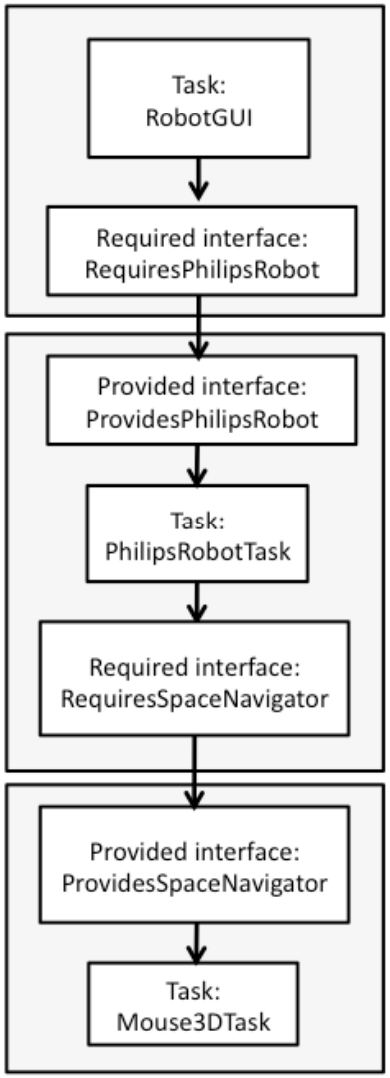


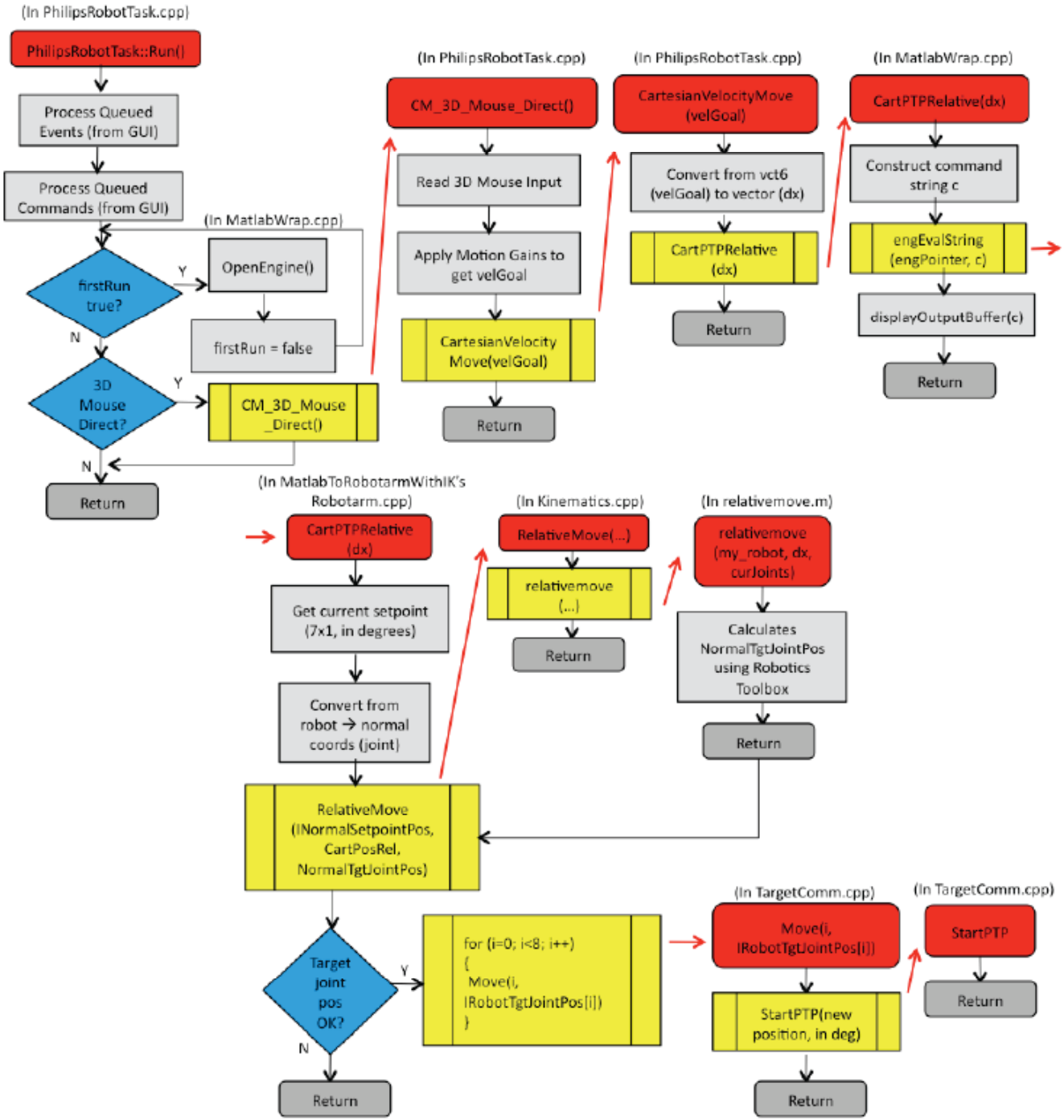
Figure 2: Multi-threaded in a single process

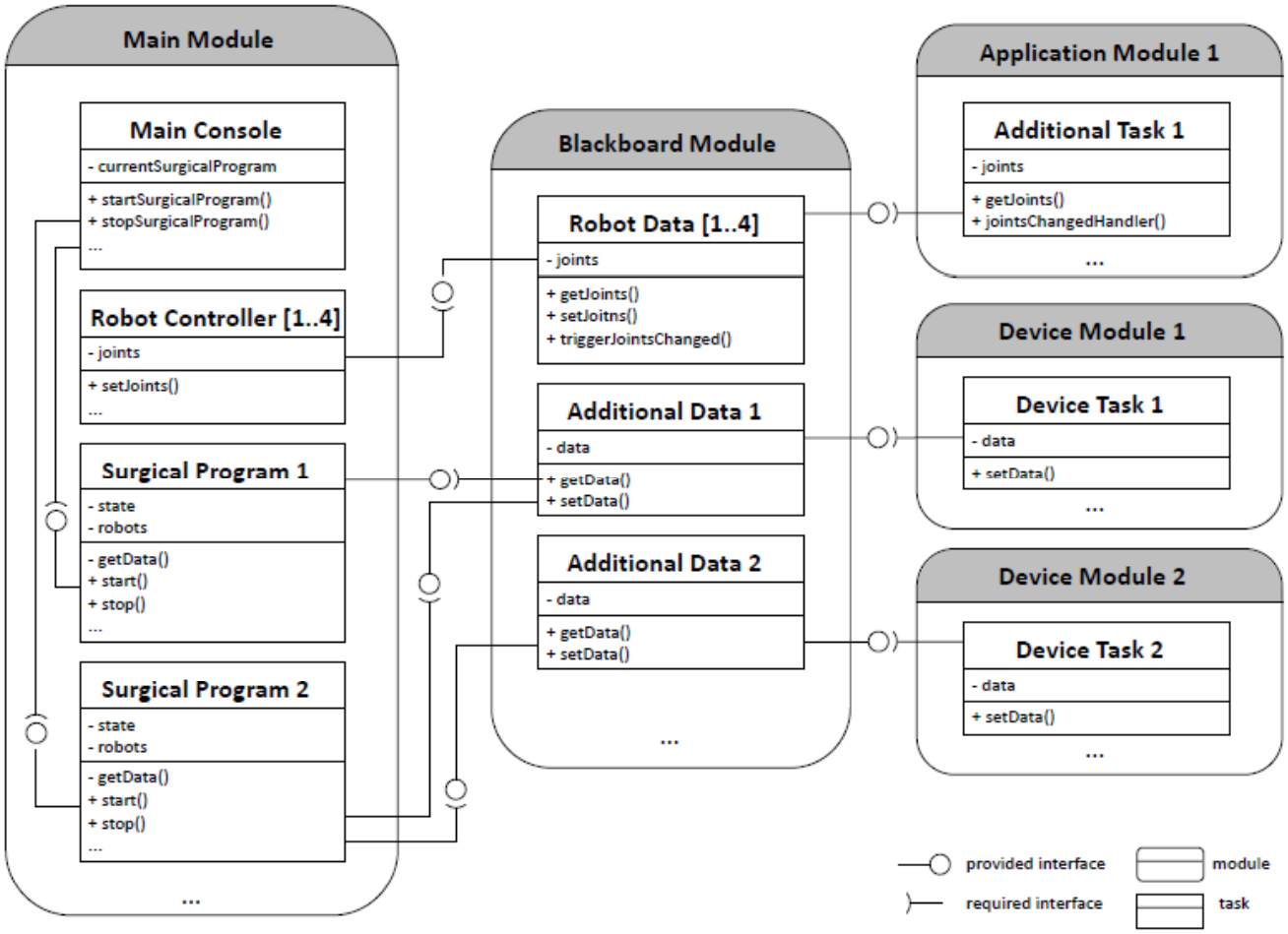


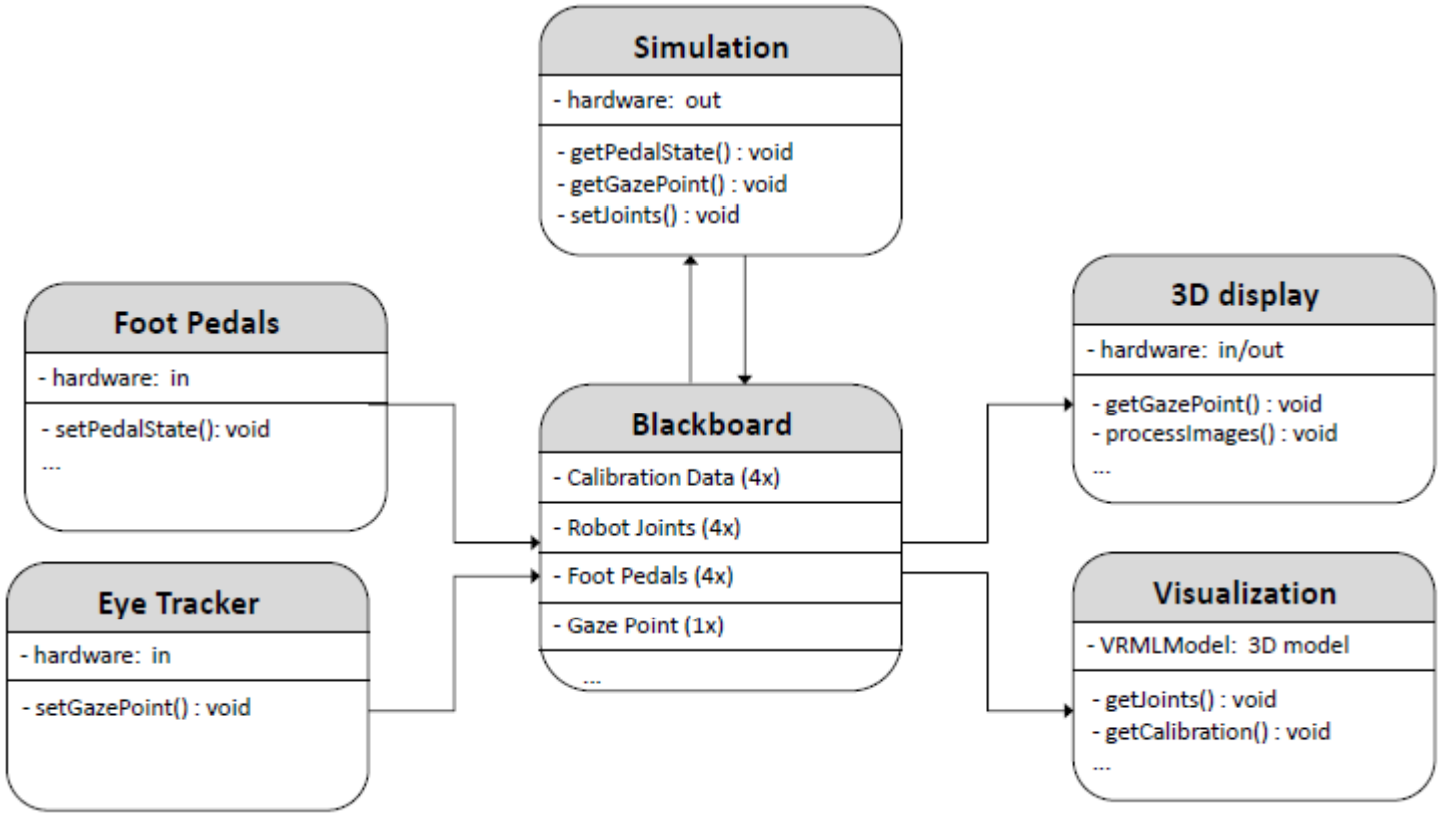

Windows System
Target System (Linux)
StereoAlgorithms
TCP Sockets
PhilipsRobot
MatlabWrap
Matlab engine
USB
Space Navigator
Matlab
Matlab + Robotics Toolbox
MatlabToRobot-armWithIK
RobotarmGUI
Servo controller application
RtMotion-USBAPI <<library>>
Kernel mode driver
USB stack
RtMotion USB Firmware <<binary>>
CORBA calls
COM calls

- Creates a pipeline




(In PhilipsRobotTask.cpp)
PhilipsRobotTask::Run()
Process Queued Events (from GUI)
Process Queued Commands (from GUI)
(In MatlabWrap.cpp)
firstRun true?
Y
OpenEngine()
N
firstRun = false
3D Mouse Direct?
Y
CM_3D_Mouse_Direct()
N
Return
(In PhilipsRobotTask.cpp)
CM_3D_Mouse_Direct()
Read 3D Mouse Input
Apply Motion Gains to get velGoal
CartesianVelocity Move(velGoal)
Return
(In PhilipsRobotTask.cpp)
CartesianVelocityMove (velGoal)
Convert from vct6 (velGoal) to vector (dx)
CartPTPRelative (dx)
Return
(In MatlabWrap.cpp)
CartPTPRelative(dx)
Construct command string c
engEvalString (engPointer, c)
displayOutputBuffer(c)
Return
(In MatlabToRobotarmWithIK's Robotarm.cpp)
CartPTPRelative (dx)
Get current setpoint (7x1, in degrees)
Convert from robot → normal coords (joint)
RelativeMove (INormalSetpointPos, CartPosRel, NormalTgtJointPos)
Target joint pos OK?
Y
for (i=0; i<8; i++)
{
Move(i, IRobotTgtJointPos[i])
}
N
Return
(In Kinematics.cpp)
RelativeMove(...)
relativemove (...)
Return
(In relativemove.m)
relativemove (my_robot, dx, curJoints)
Calculates NormalTgtJointPos using Robotics Toolbox
Return
(In TargetComm.cpp)
Move(i, IRobotTgtJointPos[i])
StartPTP(new position, in deg)
Return
(In TargetComm.cpp)
StartPTP
Return


Main Module
Main Console
- currentSurgicalProgram
+ startSurgicalProgram()
+ stopSurgicalProgram()
...
Robot Controller [1..4]
- joints
+ setJoints()
...
Surgical Program 1
- state
- robots
- getData()
+ start()
+ stop()
...
Surgical Program 2
- state
- robots
- getData()
+ start()
+ stop()
...
...
Blackboard Module
Robot Data [1..4]
- joints
+ getJoints()
+ setJoitns()
+ triggerJointsChanged()
Additional Data 1
- data
+ getData()
+ setData()
Additional Data 2
- data
+ getData()
+ setData()
...
Application Module 1
Additional Task 1
- joints
+ getJoints()
+ jointsChangedHandler()
...
Device Module 1
Device Task 1
- data
+ setData()
...
Device Module 2
Device Task 2
- data
+ setData()
...
provided interface
required interface
module
task


Simulation
- hardware: out
- getPedalState() : void
- getGazePoint() : void
- setJoints() : void
Foot Pedals
- hardware: in
- setPedalState(): void
...
Eye Tracker
- hardware: in
- setGazePoint() : void
Blackboard
- Calibration Data (4x)
- Robot Joints (4x)
- Foot Pedals (4x)
- Gaze Point (1x)
...
3D display
- hardware: in/out
- getGazePoint() : void
- processImages() : void
...
Visualization
- VRMLModel: 3D model
- getJoints() : void
- getCalibration() : void
...

Endoscope

- Gaze contingent endoscope control is one of the various options offered as part of the main module. The center of the endoscopic camera image gets automatically aligned with the surgeon's fixation point on the 3D screen, as long as a foot pedal is pressed. Consequently, two hardware components act as input (writer modules):
  - The foot pedal module reads the current state of the four pedals (pressed/released).
  - The eye tracker module processes the gaze position, obtained by the eye tracker glasses.

Endoscope - 2

- The endoscope is tracked automatically as long as the length of the vector is greater than a certain threshold. The main module directly talks via UDP to the robot's hardware controller. The robots have a clock cycle of 6.5ms, which means that in every interval at least one set of joint positions needs to arrive at the hardware controller. This timing could not be met by a separate module that reads values from the blackboard and sends them to the robotic hardware, as the calculation of the trocar kinematics already takes about half of the cycle time. Nevertheless, joint values are written to the blackboard for further consumption, e.g., by the visualization module.

Eye tracker

The hardware will be interfaced and read out in accordance with the device-specific timings. The obtained values are then published to the central storage instance, the blackboard. The eye tracker [3] is connected via FireWire to a Mac; the foot pedals are connected to the parallel port of the main PC, which is running a standard Linux. All necessary pre-processing steps, e.g., a recursive time-series filtering [6] to smooth the approximately 400 values/sec obtained by the eye tracker, are performed outside and thus relieves the main module.

Blackboard

- Besides the already mentioned data, the blackboard holds also calibration data of the surgical instruments, spatial calibration data of the robot bases, and the joint angles of each robot.

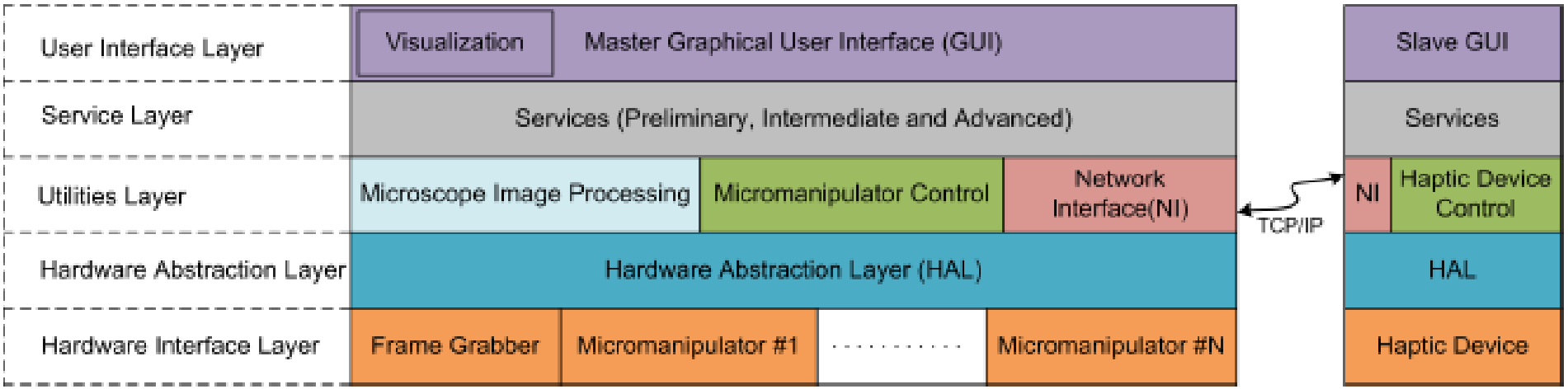
Display

- The scenario involves two modules that act as readers:
- The 3D display module acquires two video streams from the endoscopic camera. After de-interlacing, correction of brightness and size, the images are displayed at 25fps on the stereo screen. It's running on a Windows machine and also reads the current (smoothed) gaze point to visualize its position on the screen. The visual feedback to the operator improves operability.
- The visualization module shows a 3D environment of the scene, including robot and instrument movements, the operating table, and the master console. To update the joint angles of the models, this module reads the calibration data and the joints from the blackboard at 25Hz.

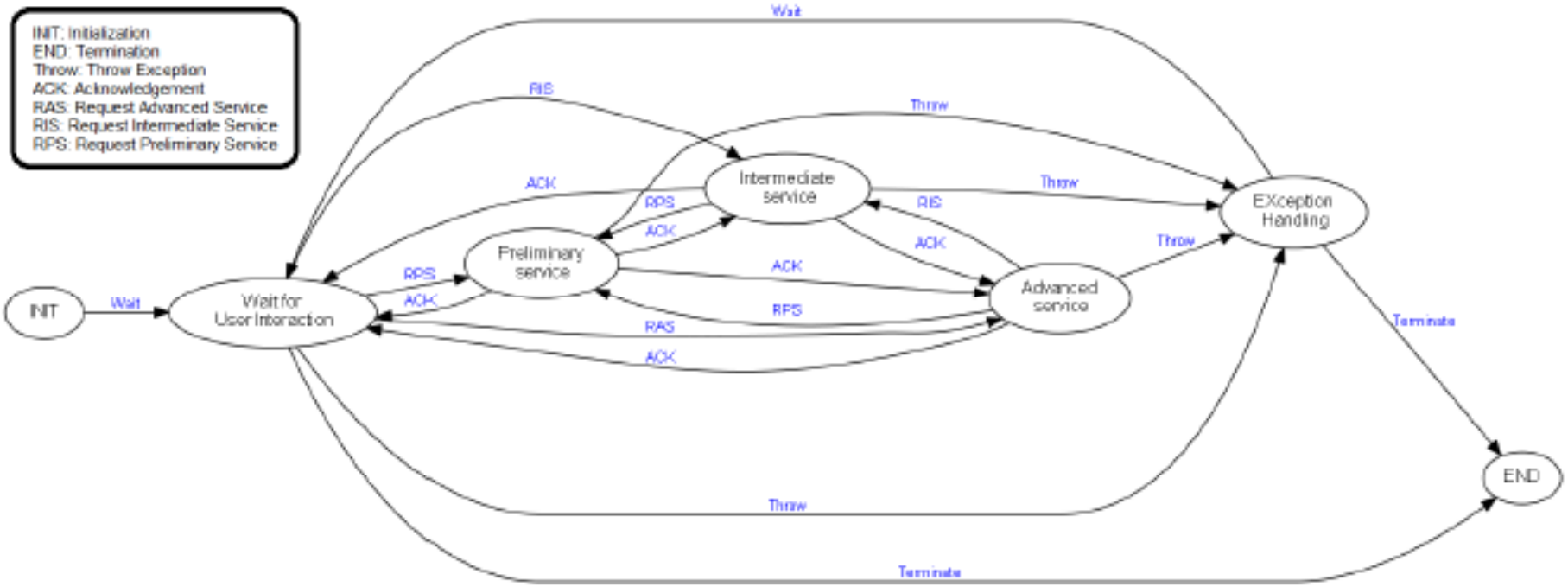
Latency

- The main module directly talks via UDP to the robot's hardware controller. The robots have a clock cycle of 6.5ms, which means that in every interval at least one set of joint positions needs to arrive at the hardware controller. This timing could not be met by a separate module that reads values from the blackboard and sends them to the robotic hardware, as the calculation of the trocar kinematics already takes about half of the cycle time. Nevertheless, joint values are written to the blackboard for further consumption, e.g., by the visualization module.

Yet another architecture



State machine



Step 4: Choose a Design Concept That Satisfies the Architectural Drivers

- Styles and patterns filtered by qualities
- When do you use …

Driver	Pattern
Efficiency	Pipe/filter
Modifiability	Layer
Flexibility	MVC
Security	Client/server

- Keep a table of these

- What technologies will/must be used
- How will the application be deployed?
- What are the crosscutting issues?

Autosar.org

http://www.autosar.org/index.php?p=3&up=1&uup=2&uuup=0

AUTOSAR_SWS_StandardTypes --- type definitions

AUTOSAR_SRS_BSWGeneral --- Requirements

AUTOSAR_TR_SafetyConceptStatusReport --- requirements for safety

AUTOSAR_TR_BSWUMLModelModelingGuide --- defines how to do UML modeling in Autosar

AUTOSAR_TR_BSWModuleList --- list of modules

AUTOSAR_MOD_BSWUMLModel --- not certain what tool is used

AUTOSAR_EXP_LayeredSoftwareArchitecture – detailed architecture

AUTOSAR_EXP_InterruptHandlingExplanation --- how interrupts are handled

AUTOSAR_EXP_ErrorDescription --- error flows

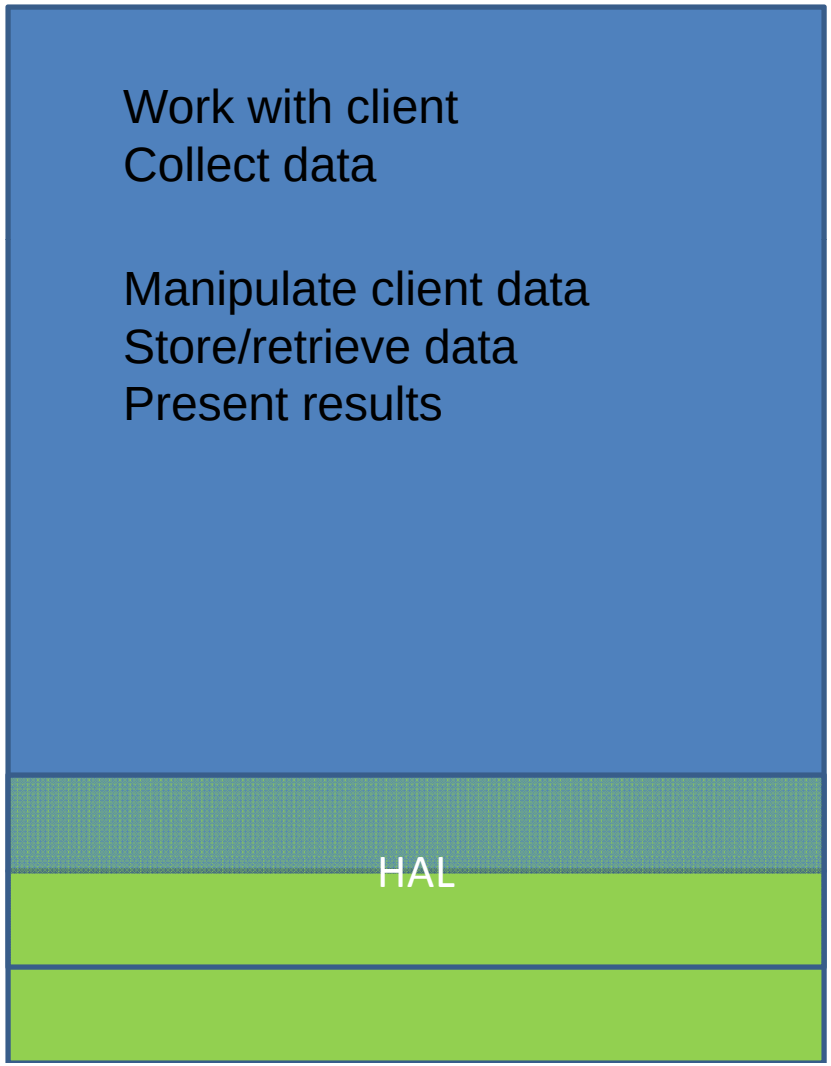
AUTOSAR_EXP_ApplicationLevelErrorHandling --- error handling in applications

Step 5: Instantiate Architectural Elements and Allocate Responsibilities

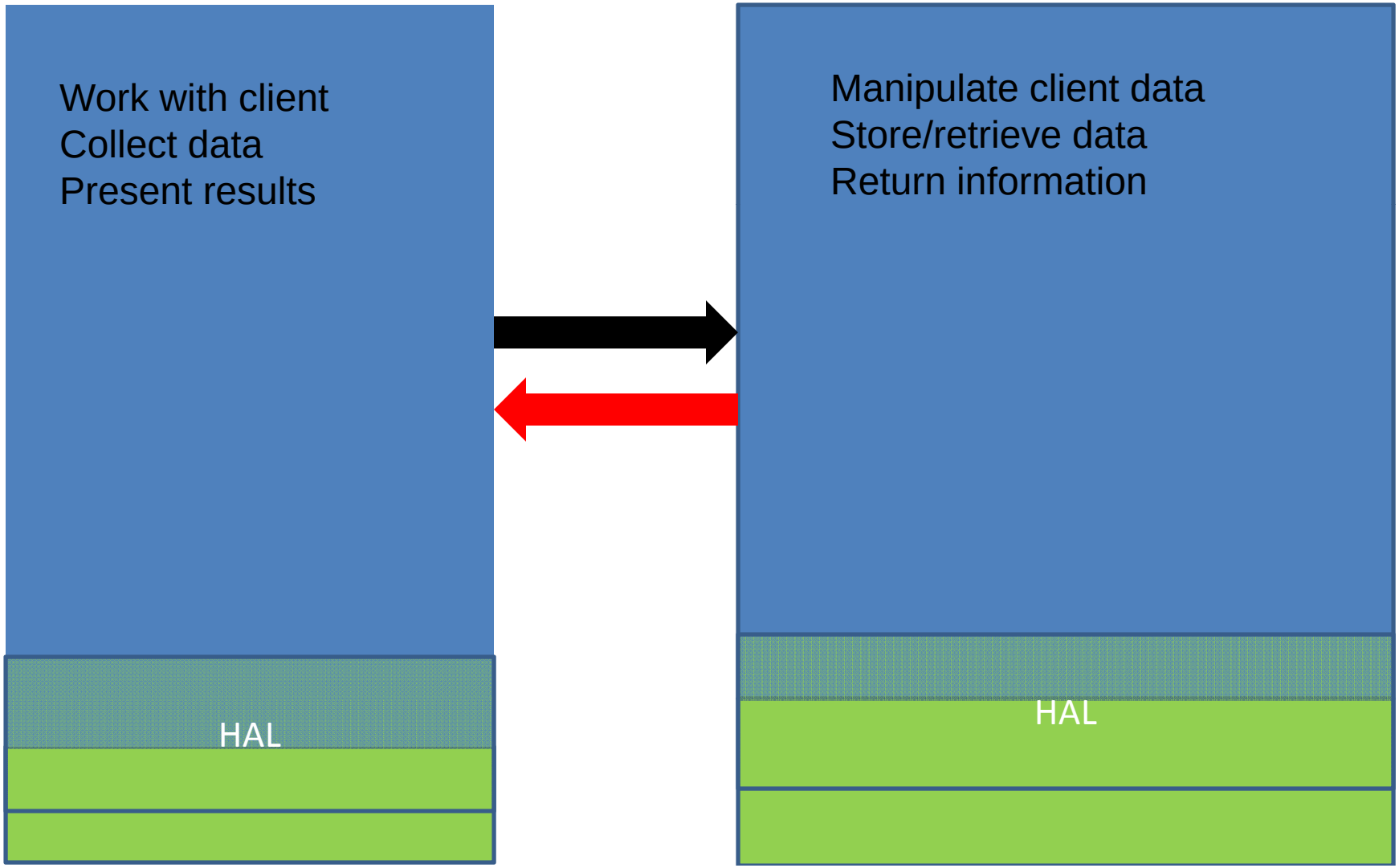
We begin with the monolith and all of the uses of the system
(Why the uses and not the requirements?)

When we decompose the monolith we also decompose the responsibilities

We also add new responsibilities from splitting some responsibilities.



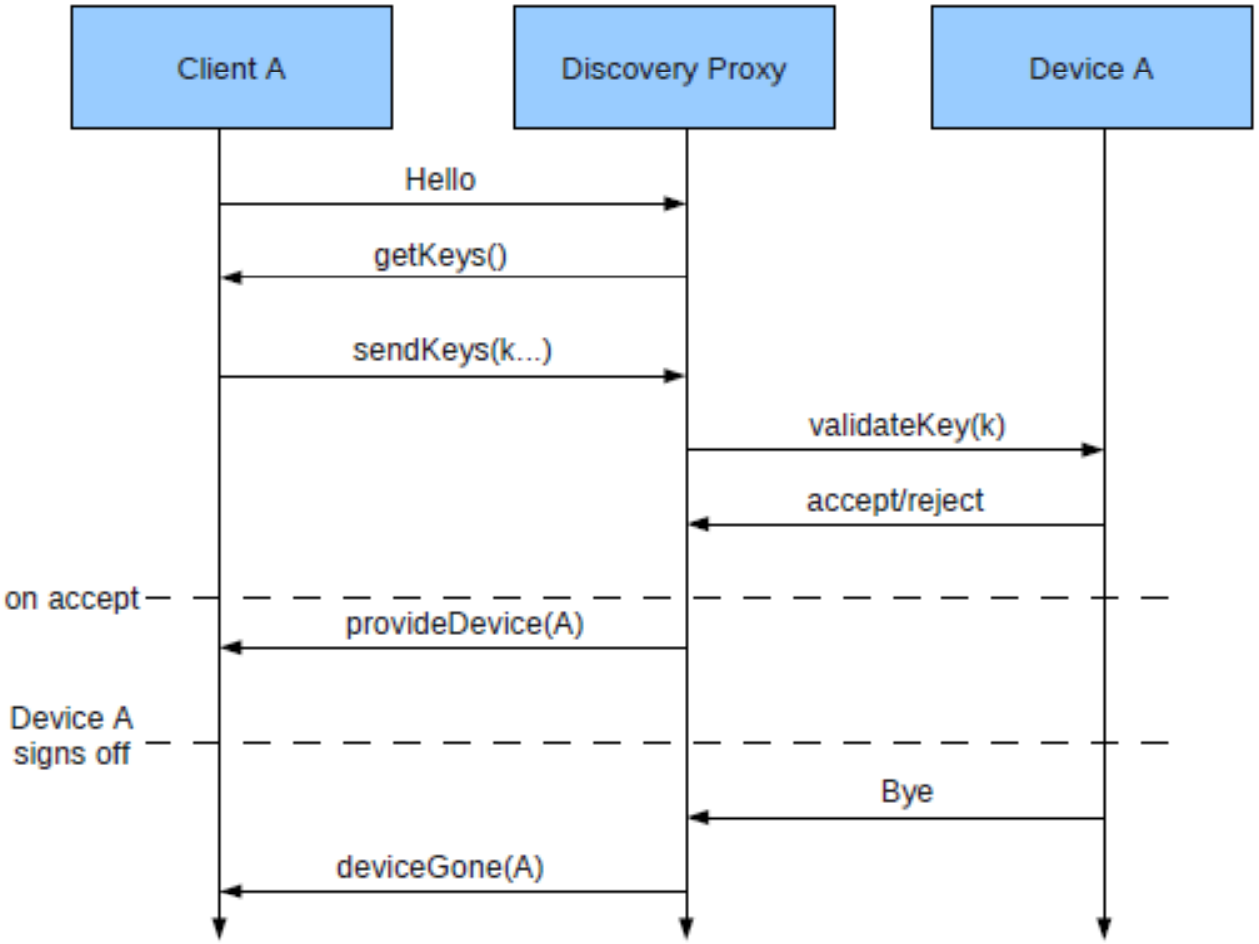
Step 5: Instantiate Architectural Elements and Allocate Responsibilities



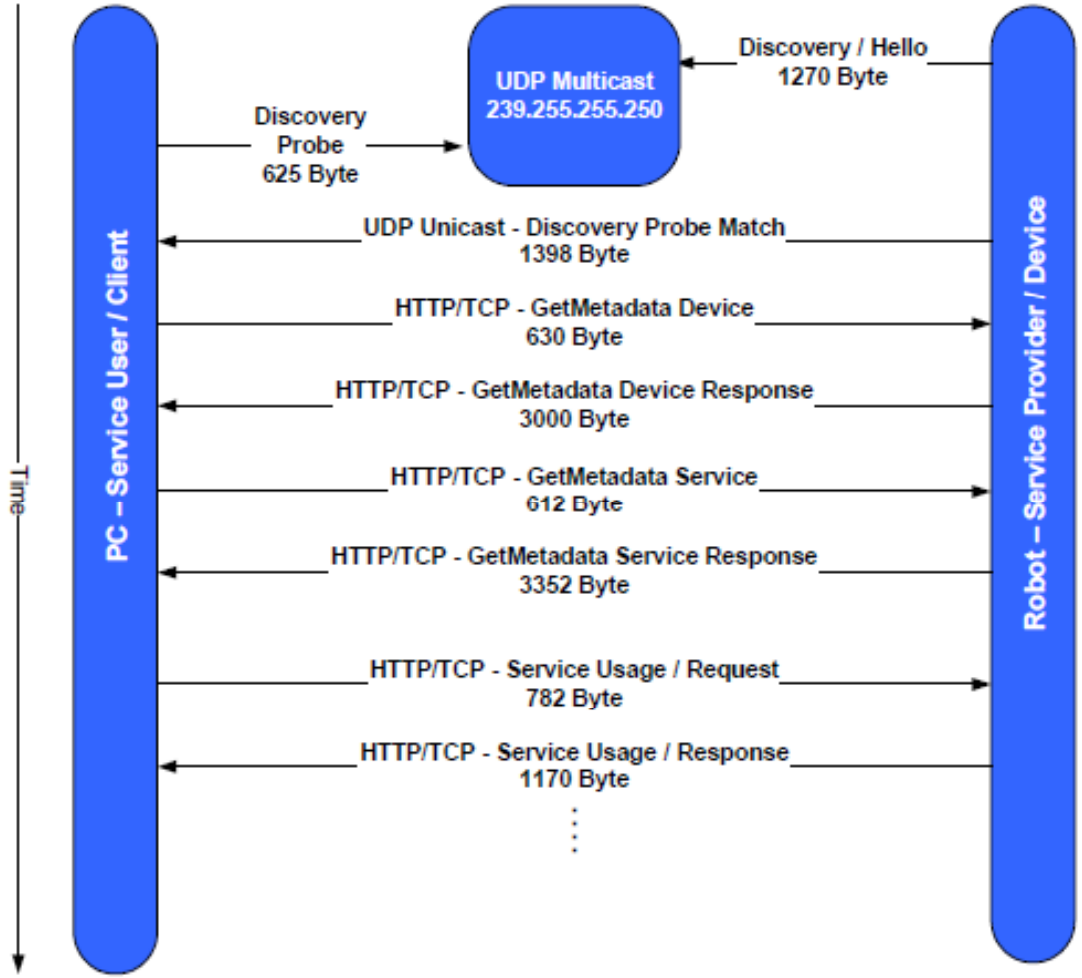
Step 6: Define Interfaces for Instantiated Elements

- Start with all the requirements
- What does each module need from others and what does it produce?
- Requires:
- Provides:

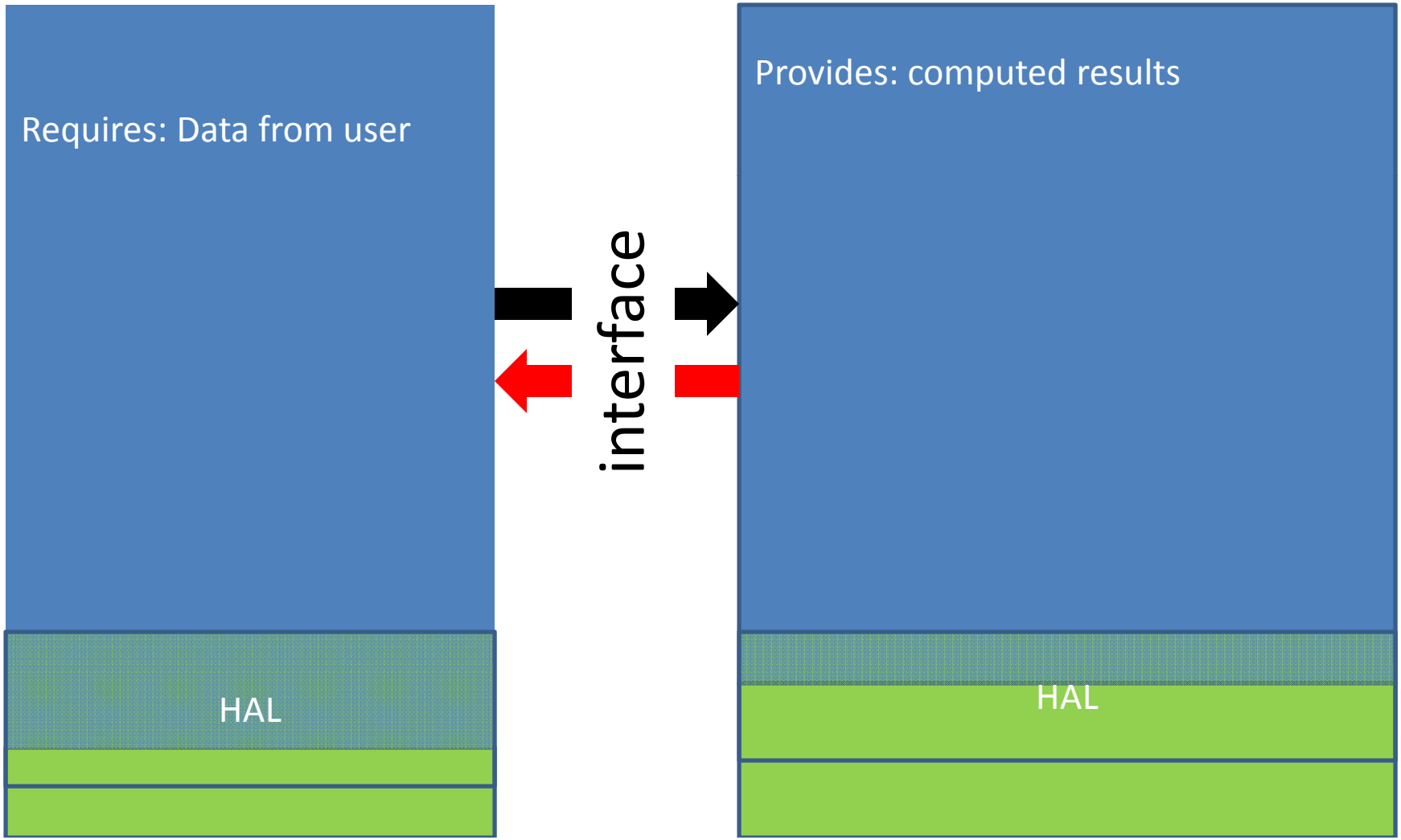
Registration



Specification



Step 6: Define Interfaces for Instantiated Elements



Here's what you're going to do

- Scan all of the Autosar descriptions
- Do a deep read of: AUTOSAR_TR_SafetyConceptStatusReport
- Expand your architecture model to include safety aspects
- Give specifications for the modules in your architecture
- Read http://www.autosar.org/download/R4.0/AUTOSAR_SRS_ModeManagement.pdf
- Use modes in either one module of your architecture or as an overall element
- Submit the *.aadl files by Monday Feb 11th by 11:59pm.