A Layered Approach for Identifying Systematic Faults of Component-based Software Systems

Min Yang Jung

Ph.D. Student
Department of Computer Science
Johns Hopkins University

June 20, 2011
Outline

• Problem
• cisst Component Model and Architectures
• Layered Approach
• Fault Model of Component-based Software Systems (CBSS)
• Illustrative Example: Robot Control Architecture
• Conclusions
Problem

- Increasing complexity and scale of modern software systems: component-based software engineering (CBSE)
- Robotics: one of representative domains
- Medical/surgical robotics: safety
Medical Robot Systems at JHU


ROBODOC® System for Orthopaedic Surgery (joint project with Curexo Technology Corporation)
Problem

• Increasing complexity and scale of modern software systems: component-based software engineering (CBSE)

• Robotics: one of representative domains

• Medical/surgical robotics: safety

**Problem**  How to make “safe” component-based software systems for medical and surgical robots?

**Specific Problem** (as a starting point) fault identification of component-based software systems (CBSS)
Problem

• “How to make safe component-based software systems for medical and surgical robots?”
**cisst Component Model**

- *cisst* base component, *mtsComponent* (cisst: Computer-Integrated Surgical Systems and Technology)

![Diagram of cisst Component Model]

- Provided Interfaces: Commands, Event Generators
- Input Interfaces: Buffer
- Required Interfaces: Functions, Event Handlers
- Output Interfaces: Component state
cisst Component Model

- Structure and connection of client and server components:
**cisst** Component-based System

- Developed and maintained by JHU
- Open-source, C++
- Cross platform: Windows, Linux, MAC OS X, real-time Linux variants (e.g., RTAI, Xenomai), QNX

**System architecture**
- Multi-threaded architecture
- Multi-process (distributed) architecture

> **Same programming model**: Seamless and flexible deployment of components regardless of architecture
**cisst System Architecture**

- Multi-threaded architecture

![Diagram showing the system architecture with components and interfaces.]

- Process boundary
- Port for internal interfaces
- Port for user (or application) interfaces

Min Yang Jung, WCOP'11, Boulder, CO
**cisst System Architecture**

- Multi-process (*distributed*) architecture

![Diagram of cisst System Architecture]

- **Process boundary**
- **Port for internal interfaces**
- **Port for user (or application) defined interfaces**
- **Local connection**
- **Remote connection**
Approach

- Proposed approach: A layered approach to identify systematic faults of component-based software systems (CBSS)
Approach

• Proposed approach: A *layered* approach to identify *systematic* faults of component-based software systems (CBSS)
Approach

• Proposed approach: A layered approach to identify systematic faults of component-based software systems (CBSS)

“Layered” approach:
Decompose system into five layers
(System, Process, Component, Interface, Execution)
Approach

• Proposed approach: A \textit{layered} approach to identify \textbf{systematic} faults of component-based software systems (CBSS)

\textbf{“Systematic” faults:}

• Domain or application independent
• Inherent to underlying framework
• Include non-functional aspects
• Proposed approach: A *layered* approach to identify *systematic* faults of *component-based* software systems (CBSS)
Approach

- Five hierarchical layers:
  1. System
  2. Process
  3. Component
  4. Interface
  5. Execution
1. **System Layer**
1. **System Layer**

<table>
<thead>
<tr>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process</strong></td>
<td>- Process faults (e.g., crash, hang)</td>
</tr>
<tr>
<td><strong>Connection</strong> (internal)**</td>
<td>- Connection faults (e.g., disconnection)</td>
</tr>
<tr>
<td><strong>(sub-layer)</strong></td>
<td>- Network faults (e.g., exception)</td>
</tr>
<tr>
<td></td>
<td>- Faults from <em>Process</em> layer</td>
</tr>
</tbody>
</table>

- Process boundary
- Remote connection
- Local connection
- Port for internal interfaces

(a) Single process setup
(b) Multi-process setup
2. Process Layer
2. **Process Layer**

<table>
<thead>
<tr>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>▪ Component faults (e.g., unexpected state)</td>
</tr>
<tr>
<td>(sub-layer)</td>
<td>▪ Faults from <em>Component</em> layer</td>
</tr>
</tbody>
</table>
3. Component Layer
3. Component Layer

<table>
<thead>
<tr>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context integrity</td>
<td>▪ Structural (functional) faults (e.g., periodicity violation)</td>
</tr>
<tr>
<td></td>
<td>▪ Non-structural (non-functional) faults (e.g., performance; application-specific)</td>
</tr>
<tr>
<td>(sub-layer)</td>
<td>▪ Faults from Interface layer</td>
</tr>
</tbody>
</table>

Port for internal interfaces
Port for user (or application) defined interfaces

Min Yang Jung, WCOP'11, Boulder, CO
4. Interface Layer
4. Interface Layer

(a) Interfaces in the same process

(b) Interfaces in different processes

<table>
<thead>
<tr>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>▪ On-connect faults (e.g., interface specification mismatch)</td>
</tr>
<tr>
<td>(sub-layer)</td>
<td>▪ Post-connect faults (e.g., network issues)</td>
</tr>
<tr>
<td></td>
<td>▪ Faults from <em>Execution</em> layer</td>
</tr>
</tbody>
</table>
5. Execution Layer
5. Execution Layer

Elements

- Command object
- Function object
- Event generator
- Event handler

Faults

- Execution inefficiency
- Network faults
- Invalid payload faults

Local connection

Remote connection
Fault Model of CBSS

Table 1: Model of systematic faults of component-based software systems

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Process, Connection (internal)</td>
<td>Process faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connection faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Process layer</td>
</tr>
<tr>
<td>Process</td>
<td>Component</td>
<td>Component faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Component layer</td>
</tr>
<tr>
<td>Component</td>
<td>Context integrity</td>
<td>Structural faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-structural faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Interface layer</td>
</tr>
<tr>
<td>Interface</td>
<td>Connection</td>
<td>On-connect faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post-connect faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Execution layer</td>
</tr>
<tr>
<td>Execution</td>
<td>Command object, Function object, Event generator, Event handler</td>
<td>Execution inefficiency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invalid payload faults</td>
</tr>
</tbody>
</table>
Robot Control Architecture

Robot with Camera

Application

Computer

Sensor
- Joint Encoder
- Video Reader

Controller

Console

Control Process

Operator

Human

Effector
- Actuator

Visualization Process

Visualizer
- GUI

cisst

cisst

Process boundary
Local connection
Remote connection

Min Yang Jung, WCOP'11, Boulder, CO
Robot Control Architecture

Robot with Camera

Application

Robot with Camera

Computer

Sensor
<<component>>
Joint Encoder

Sensor
<<component>>
Video Reader

Effector
<<component>>
Actuator

Control Process

Controller

Console

Visualizer

GUI

Visualization Process

cisst

cisst

Process boundary
Local connection
Remote connection

Min Yang Jung, WCOP'11, Boulder, CO
Robot Control Architecture

Robot with Camera Control Process

Controller

Console

Visualization Process

Human

Application

Robot with Camera

Computer

Sensor

<<component>>
Joint Encoder

<<component>>
Video Reader

Effector

<<component>>
Actuator

Visualization

<<component>>
Visualizer

<<component>>
GUI

GCM Process

<<component>>
GCM

LCM

cisst

Human

Min Yang Jung, WCOP'11, Boulder, CO
Application of Fault Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process, Connection (internal)</td>
<td>Process faults, Connection faults, Network faults, Faults from Process layer</td>
</tr>
</tbody>
</table>

Min Yang Jung, WCOP'11, Boulder, CO
Application of Fault Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Component</td>
<td>Component faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Component layer</td>
</tr>
</tbody>
</table>
Application of Fault Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Context integrity</td>
<td>Structural faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-structural faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from Interface layer</td>
</tr>
</tbody>
</table>
Application of Fault Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>Connection</td>
<td>On-connect faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post-connect faults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faults from <em>Execution</em> layer</td>
</tr>
</tbody>
</table>
Application of Fault Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Elements</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>Command object, Function object,</td>
<td>Execution inefficiency,</td>
</tr>
<tr>
<td></td>
<td>Event generator, Event handler</td>
<td>Network faults, Invalid payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>payload faults</td>
</tr>
</tbody>
</table>

Min Yang Jung, WCOP'11, Boulder, CO
Application of Fault Model
Conclusions

• A layered approach has been proposed to identify systematic faults of CBSS.
• System designers can benefit from the model at three different phases
  – *Design phase*: Provides structured tool or guideline to systematically analyze the system
  – *Implementation phase*: Reduces overhead for fault handling at user(application) level
  – *Analysis phase*: Can be extended to automatic fault analysis, automatic generation of test cases, or system safety evaluation
Future Works

• Improve the proposed fault model (generality, flexibility)
  – Apply the fault model to other component models

• Implement the fault model using the *cisst* framework (e.g., fault detection)
  – (Experimentally) validate the effectiveness of the fault model

• Extend the fault model
  – Can be used for fault-tolerant or safety-critical systems research
Acknowledgements

• This work is supported in part by
  – National Science Foundation (NSF) EEC 9731748, EEC 0646678, MRI 0722943
  – JHU internal funds

Dr. Peter Kazanzides (advisor)

Dr. Russell H. Taylor
(director of ERC CISST)
Thank You

http://www.cisst.org/cisst

cisst libraries

News

- **Oct. 4, 2010**: We made the entire SVN repository public today. This included some reorganization of the repository, so it may be necessary to use the SVN switch command.
  - If currently using `https://svn.lcsr.jhu.edu/cisst/branches/main`, switch to `https://svn.lcsr.jhu.edu/cisst/trunk`
  - If currently using `https://svn.lcsr.jhu.edu/cisst/public/trunk`, switch to `https://svn.lcsr.jhu.edu/cisst/trunk` (or, it may be simpler to just delete your current working copy and do a new checkout)

Overview

The cisst package is a collection of libraries designed to ease the development of computer assisted intervention systems. One motivation is the development of a Surgical Assistant Workstation (SAW), which is a platform that combines robotics, stereo vision, and intraoperative imaging (e.g., ultrasound) to enhance a surgeon's capabilities for minimally invasive surgery (MIS). All software is available under an open source license, which can be found [here](https://svn.lcsr.jhu.edu/cisst). The SVN repository contains a public trunk and development branches. We plan to move all software to the public trunk when it is well documented and tested. If you would like access to the development branches, please contact us. The following table describes the cisst libraries and indicates whether they are currently available in the public trunk or the development branch.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>cisstCommon</td>
<td>Common infrastructure such as logging, class and object registries, serialization and de-serialization, etc.</td>
<td>Public</td>
</tr>
<tr>
<td>cisstVector</td>
<td>Basic linear algebra and spatial transformations in two dimensions (2D) and three dimensions (3D)</td>
<td>Public</td>
</tr>
<tr>
<td>cisstNumerical</td>
<td>Thread-safe numerical methods (relies on LAPACK3E)</td>
<td>Public</td>
</tr>
<tr>
<td>cisstInteractive</td>
<td>C++ classes and Python scripts for the Interactive Research Environment (IRE)</td>
<td>Public</td>
</tr>
<tr>
<td>cisstOSAbstraction</td>
<td>Operating system services (e.g., threads, mutual exclusion, etc.) for Windows, Linux, Mac OS X, RTAI/Linux, ...</td>
<td>Public</td>
</tr>
<tr>
<td>cisstMultiTask</td>
<td>Component-based framework for defining tasks and devices, provided and required interfaces, and command objects</td>
<td>Public</td>
</tr>
</tbody>
</table>
Thank You

• Questions and feedbacks?