

A. Project Summary

Project Title	Networked Robots
Time-frame	Spring 2009 (1/26/2009 – 5/14/2009)
Summary	The current implementation of the CISST package—specifically, the <code>cisstMultiTask</code> library—allows multi-threaded design within a single process. As an extension of it, a solution for inter-process communication (IPC) over networks should be implemented in order to support distributed system more effectively at the library level instead of the application level.
Team Member	Min Yang Jung
Mentor	Dr. Russell Taylor, Dr. Peter Kazanzides, Anton Deguet
Date Prepared	February 17, 2009

B. Relevance and Importance

Recent technological advances in network communication have continually broadened the research area of telesurgery and teleoperation. Such area includes research performed in a single room with two local computers connected via LAN (a few meters away) and large scale research done across networks (Internet) spanning more than several thousand kilometers. From the systematic point of view, the core concept for such research is inter-process communication (IPC).

The current CISST library, however, does not support such distributed systems at the library level; it has been designed to work within a single process. Thus, a researcher who studies telesurgery or teleoperation using the CISST package would have to implement their own IPC module at the application level in order to link their software over networks. This acts as implementation overhead to a researcher, which might result in lower productivity.

Therefore, this project aims to develop a systematic solution at the library level that will allow CISST-based applications to communicate each other over networks. With this solution, we expect that distributed systems can be implemented more easily and efficiently while minimizing researcher's implementation overhead.

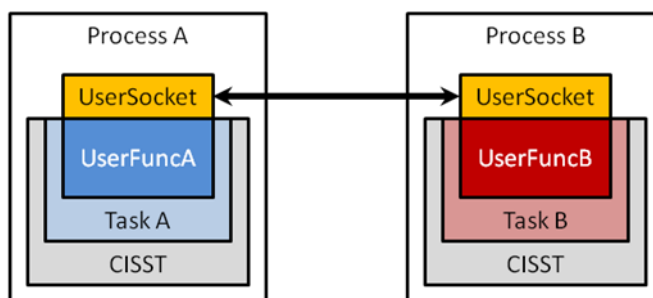
C. Technical Summary of Approach

Two major features will be developed in this project: IPC module and transparency.

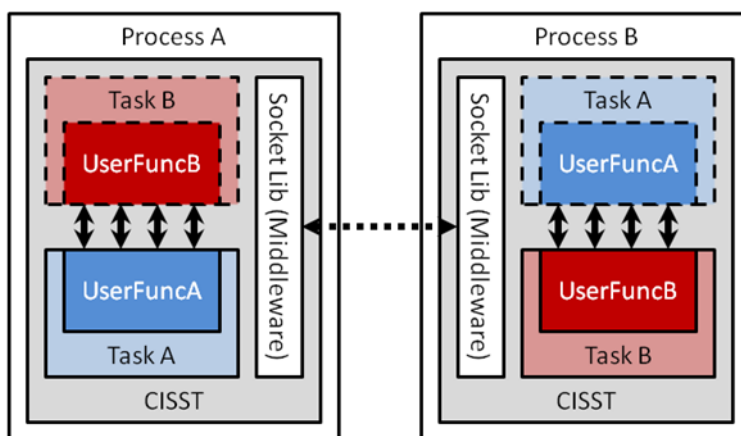
The IPC module can be implemented either building our own socket library or integrating 3rd-party networking package with the CISST library. Once it is embedded in CISST, users don't have to implement their own socket handlers any more.

The other is transparency. When two tasks on two different processes need to communicate with each other, the CISST library will set up a task proxy at the other side which acts as a delegate (Proxy pattern). In this way, a task can talk to the other task as if both tasks were on the same process (or machine). That is, all tasks can work together regardless whether they are in the same process or not.

These features are conceptually shown in the following figures.



(A) **Current architecture of an application based on the CISST library.** *UserSocket* library to handle sockets directly through network needs to be implemented at an application level.



(B) **New architecture based on proxy pattern that supports IPC.** The socket library is embedded in the CISST library – no need of *UserSocket* as in (a). Tasks can work together as if they were in the same process.

D. Deliverables

Deliverables	
Form of Deliverables	<ul style="list-style-type: none"> Software library providing links to other tasks across network Documented code
Features	
Minimum	<ul style="list-style-type: none"> Any type for which serialization already exists (e.g., primitive type such as int, double, string) Example software: two tasks of two processes over networks, command line controller
Expected	<ul style="list-style-type: none"> Global task manager (multi-cast feature supported) Basic serialization Example software: three tasks of three processes over networks, command line controller
Maximum	<ul style="list-style-type: none"> General serialization

- Python integration
- “Real” Example: controlling a virtual (or simulated) robot, Python interface (IRE) to connect and control them

E. Time Line

Phase	Month	2			3				4			
	Week	2	3	4	1	2	3	4	1	2	3	4
Min	<i>Minimum (6 weeks)</i>											
	Background/Package research											
	Proxy implementation											
	Example 1 implementation											
	Test & Debug											
Exp	<i>Expected (3 weeks)</i>											
	Global task manager implementation											
	Basic Serialization implementation											
	Example 2 implementation											
	Test & Debug											
Max	<i>Maximum (2 weeks)</i>											
	Python (IRE) integration											
	Example 3 implementation											
	Test & Debug											

F. Dependencies

Hardware	<ul style="list-style-type: none"> ▪ 1 Desktop, 1 Laptop (2 Laptop beyond minimum deliverables) ▪ Network accessibility (wired or wireless)
Software	<ul style="list-style-type: none"> ▪ Integrated Development Environment (IDE) : Microsoft Visual Studio 9 ▪ 3rd party middleware : ICE, Spread
Human Resource	<ul style="list-style-type: none"> ▪ Mentors’ time for help and advice ▪ Cooperation with two teleoperation project teams (for maximum deliverables)

G. Management Plan

- Regular meeting with Dr. Peter Kazanzides
- Project management with Dr. Russell Taylor
- No budget management required
- No hardware equipment needed

H. Reading List

- [Concept] **Serialization** : FAQ (concept, implementation, method, etc.)
- [Package] The Internet Communications Engine (ICE), Spread
- [Book] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.

- [Publication] Soundararajan, K, Brennan, R.W., ***A proxy design pattern to support real-time distributed control system benchmarking***, Lecture Notes in Artificial Intelligence 3593, pp. 133-143
- [Publication] Shapiro Marc, ***Structure and Encapsulation in Distributed Systems: The Proxy Principle***, Proceedings, International Conference on Distributed Computing Systems 1986, Pages 198-204