論文 / 著書情報
Article / Book Information

Title	Proposal of Intelligent Cross-Platform Interface for Robotics Middleware
Author	Arturo E. Ceron Lopez, Edwardo F. Fukushima
Journal/Book name	Proceedings of the 2012 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, , , pp. 382-387
Issue date	2012, 5
DOI	http://dx.doi.org/10.1109/CYBER.2012.6392584
URL	http://www.ieee.org/index.html
Copyright	(c)2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Proposal of Intelligent Cross-Platform Interface for Robotics Middleware

Arturo E. Cerón López and Edwardo F. Fukushima
Department of Mechanical and Aerospace Engineering
Tokyo Institute of Technology
Tokyo, Japan
aceron@robotics.mes.titech.ac.jp, fukusima@mes.titech.ac.jp

***Abstract*—The current standard approach for developing robot software applications is to use middleware for robotics. Protocols and paradigms have been set by the available middleware. However, not all of them share a common interface, resulting in difficulties when developing robot applications using software developed for different platforms. Even inside the same platform similar issues arise when software elements inside it are made by different developers. The research objective is to propose and define an Intelligent Cross-Platform Interface (ICPI) that enables data sharing among software elements from different developers, as well as data from different middleware platforms. Such interface is to be complemented by additional modules residing inside it to enhance the functionality. In this article one of the modules is discussed, this module is for the administration of data and software in a robot application. The Administration by Roles module is presented to act as the manager module of the ICPI, the module enables the categorization of software elements and their related data.**

Keywords-middleware; robotics; interface; cross-platform; administration

I. INTRODUCTION

Nowadays, development of robot software has become more accessible and friendlier for the user. The current standard approach is to use middleware for robotics [1][2], which is composed of a set of services that allow the interaction of specialized robot software elements, each one performing a specific task. Some of those elements may deal with information related to a physical device (e.g. cameras, microphones and motors) and some others are data processing functions (e.g. object recognition, speech processing and inverse kinematics); all together can build up a more complex robot application. Each element runs as an independent process, while they transfer information across a network (which can be internal or external) to share data with other available elements.

Various middleware platforms for robotics have been developed in the recent years [3]~[10]; all of them provide tools for the developers to create complex robotics applications in a simpler way. There are many middleware platforms being utilized among users, researchers and research groups; some of them are the following: Robot Operating System (ROS) [3][4], Yet Another Robot Platform (YARP) [5][6], Microsoft Robotics Developer Studio ® (MRDS) [7][8], National Instruments LabVIEW ® (LV) [9] and others using the CORBA standard by OMG [11], like the Robotics Technology Middleware (RTM) [10].

Different protocols and paradigms have already been set by each one of them. However, not all of them share a common interface, resulting in some difficulties for the users like the transmission of data among platforms.

Since a robot application requires of great amount of code, the purpose of developing middleware for robotics is to make robot software as modular and reusable as possible. In this approach, it is common to see software elements performing highly specialized functions; these elements can be connected in many ways to other ones in order to transfer data and share services between them. Data types have also been defined, each platform having the basic well known types, plus some other defined for use in a specific platform.

While platforms can reuse the developed code and data transmitted inside them, often the user cannot reuse code and/or share data across platforms (Fig. 1).

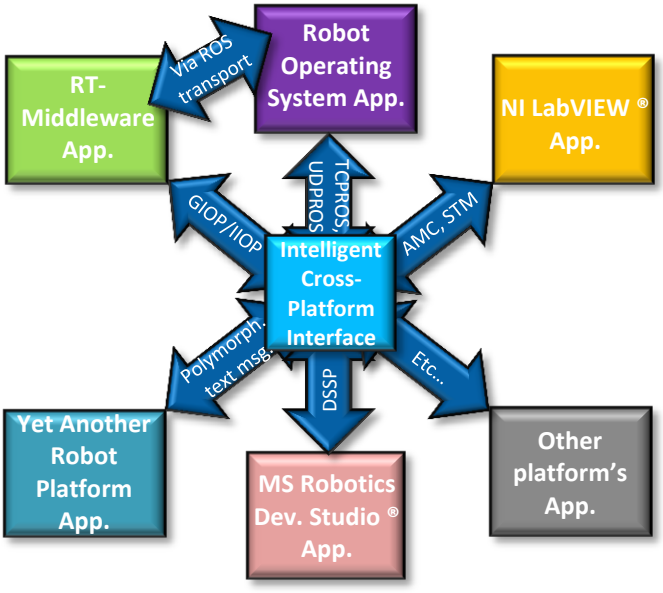

Figure 1. General concept for the connection among multiple robotics middleware platforms using the proposed ICPI.

Each of the middleware platforms has its own characteristics and advantages; also some software elements are only available in few platforms and/or are more stable in some platforms than in others. It is difficult to conceive a common robotics middleware platform since needs, problems and points of view are different among communities [2]. This is leading to a phenomenon in robotics software application development; it is that the user and researcher communities are becoming segmented. People belonging to each segment are using the same platform among them and attach to it because they get familiar with the way of use and development of applications.

However, it becomes troublesome when a link between different platforms is required for sharing their characteristics and advantages. Here a series of issues regarding this point is presented:

- Every platform has defined communication protocols that lie in the application layer or between the application and transport layers. It is required to write new code to parse data between one middleware platform and another every time the user or developer wants to have a connection between them and/or with other interfaces.
- While one middleware platform may be suitable and intuitive for a group of users, another one might not. Therefore, a problem exists for finding a common paradigm that fits all the needs. Changing between platforms and developing code that help to link between them can become time consuming since some learning and training stage is needed.
- Other issues may include the way data is classified and interpreted, for example when one wants to use different software elements made by different developers; it is sometimes found that the data connections are not compatible and additional code needs to be programmed.

This series of issues are present in the currently available middleware for robotics, and they can become a problem for the user and the developer while making and testing a complex robot application. For these reasons, a method that can enable the linkage among software elements from different developers and/or from different middleware platforms is desired.

II. Introduction to Intelligent Cross-Platform Interface

This interface is composed of servers and clients, which allow data sharing and reutilization of software made by various developers for different middleware platforms, in an intuitive way for both the developer and the user. Common data structures, formats and communication protocols above the transport layer are to be set. Tools for the transition among different software elements and platforms are to be provided.

The basic features in the proposed interface are:

- Cross-platform data parsing.
- Data probing.
- Definition of data labels and unit systems.
- Data and software categorization
- Automatic management of data and software.
- Expressed software and data relationships.
- Common data structure and protocols.
- Self-maintenance of robot applications
- Graphical User Interface with virtual representation of software elements.
- Modularity for its core components.

A. *General Architecture*

The Intelligent Cross-Platform Interface (ICPI) is illustrated in Fig. 2 and shows a general architecture for it. The solid line connections represent the adopted protocol by each of the middleware platforms, while the dotted line connections show the common protocol proposed for the ICPI, which is to be defined in future works.

- ICPI Client:

In order to exchange data and features across the middleware platforms that are being used, the ICPI Client is in charge of making a link between software elements inside a middleware platform and the ICPI Server. For example, data can come either from an interface or module outside the platform in use and sent to the corresponding software elements inside the platform, but also data can be transmitted from the platform's software elements to the interfaces/modules outside the platform; additionally with the help of the ICPI Server modules, data can be transmitted from one software element in one platform to another software element in another platform. The ICPI Client is implemented according to the middleware platform's given paradigm; it also performs basic management functions related to the platform being used.

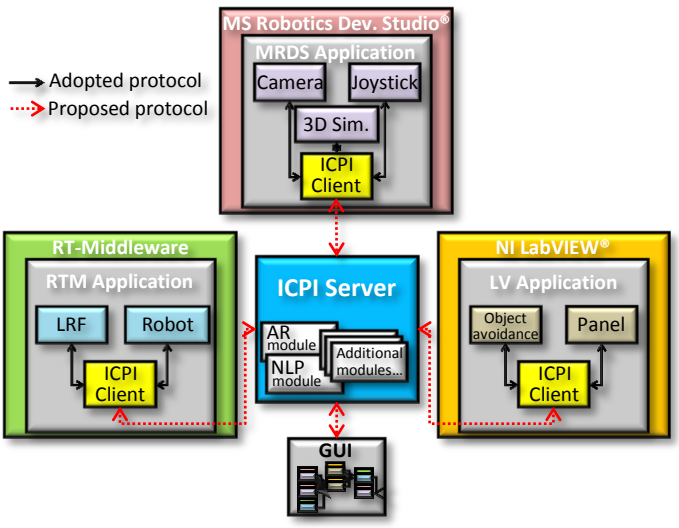

Figure 2. Robot applications in different platforms linked by the Intelligent Cross-Platform Interface (ICPI).

To achieve this, an ICPI Client is to be developed for each specific middleware platform by using the available libraries that give access to pertinent middleware platform features, as well as making use of the platform's adopted communication protocol. With this, the ICPI Client resides inside the middleware platform and acts as an element of it, while making a link to the outside.

- ICPI Server:

The ICPI Server is a stand-alone entity that unifies middleware platforms for robotics, by allowing data sharing and enhancing the functionality of the existent platforms. It is composed of a set of modules that can access data and functions from a middleware platform throughout the ICPI Clients. It has additional modules to increase its functionality and to give a more intuitive way of use; the following modules are to be implemented inside the ICPI Server architecture:

The manager module: This module is for administrating data, software elements and robot applications using defined roles.

Natural Language Processing (NLP) module: A NLP engine having a common dictionary for data labels and defined roles, and a common library for known software elements and complete robot applications.

In this article the manager module, which is part of the ICPI Server, is introduced as the Administration by Roles (AR) module.

B. Administration by Roles module:

The purpose of this module is to enable data sharing between software elements in a categorized way, while relating data with software elements and/or applications; with this, the features of the software elements can be managed in such a way that the automated building of complete applications based on the running software elements/applications and the data available in the system becomes possible.

While the middleware platforms are good at reusing code by just changing the connections between software elements, data is not always explicitly labeled and categorized (Fig. 3) and software elements do not always have a defined explicit role in the system (Fig. 4).

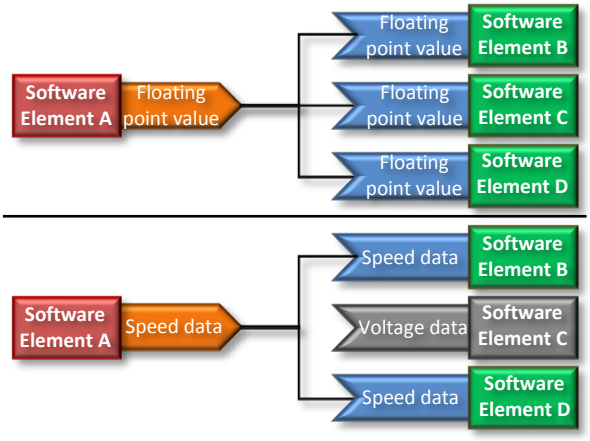

Figure 3. Example of defining labels to data for better categorization.

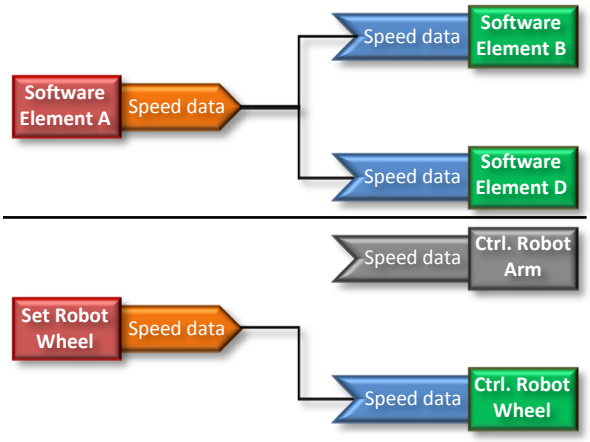

Figure 4. Defining explicit roles to the available software elements to reduce ambiguity.

As shown in the example of Fig. 3 (upper part), a source of data expressed as floating point values can be anything; theoretically this data source can be connected to any software element, having data ambiguity. However, if labels are given to data, as shown in Fig. 3 (lower part), ambiguity is reduced and connections become narrowed, while keeping data as floating point values.

In Fig. 4 (upper part), since all software elements look the same, there is still ambiguity about the role they play in the system. There is still a possibility that data connections are not correct even if data is labeled correctly. If a specific role is given to the software elements, as shown in Fig.4 (lower part), data connections can be narrowed even more, reducing the ambiguity again.

1) Labeling data in a common data structure:

A common data structure is needed to standardize the way of communicating among software elements and platforms; however, such structure must have labels describing the data contained in the structure, and labels are to be shared among platforms.

2) Explicit definition of roles:

When defining a role, one is defining the responsibility, the duty, the capability and/or an expected behavior; same concept can apply when defining roles for software elements or complete applications.

Defining a role can be as simple as stating the basic function, the data needed to perform such a function, and the data generated after performing the function; similar as in current programming languages, with the difference that this information can be shared among platforms to understand each other, the role becomes a higher level definition. It can be as specific as needed, allowing defining more complex functions and relationships of data.

3) Module implementation:

The module consists of a set of databases inside the ICPI Server for storing information about the available data, software elements and applications, as well as the following states (Fig. 5):

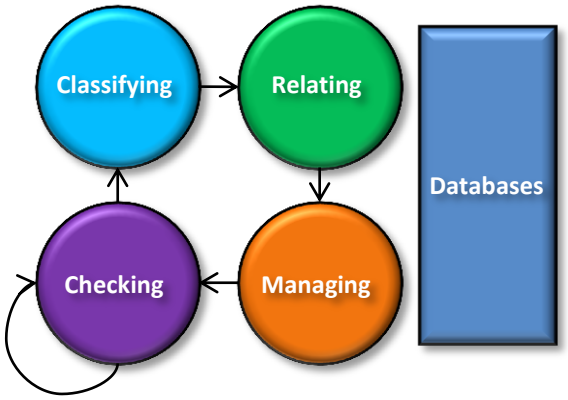

Figure 5. The Administration by Roles module.

State 0. *Checking* if a new software element and/or application is run or an existing one is terminated, if this happens, proceed to next state.

State 1. *Classifying* the available software elements and/or applications according to the information found in their defined role.

State 2. *Relating* between data and running software elements/applications.

State 3. *Managing* of the features pertinent to the software elements/applications (e.g. connections, statuses, configurations).

III. Example Implementation of the ICPI

A. ICPI Client implementation for RT-Middleware

An ICPI Client for RT-Middleware was developed using the RTC and CORBA libraries. These libraries provide means for interacting with the core architecture of the robotics middleware and the communications among software elements. This direct interaction enables the ICPI Client to behave like a virtual user, which has the freedom of utilizing the features of the platform.

The implemented ICPI Client has the capability of adding and removing ports from itself and is represented in a RTC fashion. It also can manage the features of the RTCs inside the system (e.g. configurations, connections and statutes).

B. ICPI Server implementation

As mentioned earlier, the ICPI Server is composed of many modules that enhance its functionality; the server is the one that encapsulates the modules that make it work. The implementation is made using an Administration by Roles (AR) module, which is the manager module and resides inside the ICPI Server. As a case study for this article, the AR module throughout an ICPI Client for use in the OpenRTM-aist platform will have access to the data and software elements inside this platform.

1) Implementation of the AR module

The following paragraphs describe how a prototype implementation of the AR module was developed. The basic way of operation of this module is to receive data coming from all RTCs that deliver data and send the required information to the ones that receive data. To do so, a role is defined for each RTC, as well as a fixed common data structure with labeled data.

- Defining roles for the RTCs:

In this example, it has been decided to define the roles by setting the following actions since they illustrate the most basic ones when performing a robot software function:

"Deliver": For RTCs that only deliver data.

"Receive": For RTCs that only receive data.

"Process": For RTCs that receive data in order to process it and delivers new data.

In future works, more complex actions are to be defined and later decomposed in simpler actions like the ones illustrated here.

To complement the defined role, labels are given to the related data; in this system, 3 types of data labels are defined: "Direction", "Speed" and "Control Data".

As the RTM standard gives some fields to describe each one of the RTCs, the defined role information can be placed in those fields since they are identification fields available for modification. We have chosen to put the basic action information in the *Category* field and the labels for the related data in the *Description* field.

- Labeled data structures for RTCs:

To make this case study example simple and understandable as possible, it has been decided that for the prototype AR module, data structures are to be sent and received as text, this is, character strings. Using the RTM standard, they will be of *RTC::TimedString* type for all the RTCs.

Each data structure should have a label, which describes the nature of the data inside the structure. Each structure can contain many data fields inside (e.g. "Direction" data structure can contain "X Axis", "Y Axis" and "Rotation" data fields, each one having a value) which are also labeled. The format used is the following:

<LABEL>([FLD_0:V_0][FLD_1:V_1] ... [FLD_n:V_n])

Where LABEL is the desired label for the data structure, FLD_n is the label for the n^{th} field, and V_n is the given value of the n^{th} field.

By defining data structures and labels in this way, many RTCs can complement one single data structure, for example if one RTC sends the following character string:

<DIRECTION>([X_AXIS:0.0][Y_AXIS:1.0] [ROTATION:0.5]),

and another one sends this character string:

<DIRECTION>([Z_AXIS:0.7]),

the resulting character string will be the following:

<DIRECTION>([X_AXIS:0.0][Y_AXIS:1.0][Z_AXIS:0.7] [ROTATION:0.5]),

the prototype AR module will be in charge of organizing the data provided in order to have the resulting character string as the one just shown.

It is to note that the value of each field can be an integer value, a floating point value, a text or any other customized value that can be represented in alphanumerical characters (e.g.

<DEMO_DATA>([SPEED:100.25][SENTENCE:Hello world][BUTTONS:10100011][COUNTER:4])).

- AR module states and databases:

For the implementation, the prototype AR module counts with a set of databases for the data received and the data generated (referred as the Data Structures database), also databases for classifying the RTCs according to their previously defined role. Algorithms in charge of updating the mentioned databases and of performing the described state operations are running in background as independent threads (Fig. 6).

State 0. *Checking* whether a new RTC is run or an existing one is terminated:

The AR module is always checking with the help of the ICPI Client for any change in the system that may indicate the execution or finalization of an RTC; if this happens then the next state is issued. Additionally, data inside the Data Structures database that has not been updated in a while is deleted.

State 1. *Classifying* the RTCs:

The defined roles of all RTCs are checked to see which type of action they perform, if the described action is "Deliver", the RTC is added to the "Deliver" database; if the action is "Receive", the RTC is added to the "Receive" database; if the action imply both "Deliver" and "Receive" actions, as in the case of a "Process" action, the RTC is added to both databases.

State 2. *Relating* between data and running RTCs:

Additions to each database are triggered by the ICPI Client; additions to the "Deliver" database leads to an addition of an Input Port at the ICPI Client and a port assignment for that RTC, while an addition to the "Receive" database leads to a procedure of looking for suitable data in the Data Structures database for this RTC. The ICPI Client reads the data coming from all the RTCs and passes it to the AR module, this information is added or updated in the Data Structures database, where it gets classified and ordered. Many RTCs can complement one data structure. An addition to the Data Structures database leads to an addition of an Output Port at the ICPI Client and a port assignment for that data structure, each Output Port deliver information concerning one data structure identified by the AR.

State 3. *Managing* features pertinent to the RTCs:

In this prototype AR module, the automatic activation and connection of the RTCs are the managed features; this is made

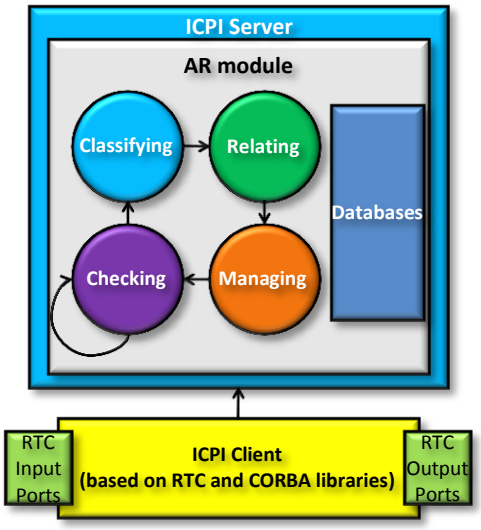

Figure 6. Interface for Administration by Roles and ICPI Client designed for RT-Middleware.

throughout the ICPI Client. Then, the RTCs that are registered in the "Deliver" database are connected to their assigned Input Ports at the ICPI Client. In the case of the RTCs registered in the "Receive" database, if there is a suitable data structure for them, the RTCs are connected to the ICPI Client at the previously assigned Output Port of the suitable data structure. Finally, when connections were successfully made, the AR module issues the activation of the RTCs via the ICPI Client.

IV. Testing the ICPI Server's AR module and the ICPI Client

To test the prototype AR module and the given ICPI Client, a simple robot application using an omnidirectional robot platform (Fig. 7), a joystick, and a simple function that given a scalar "speed" value, multiplies it with the "direction" vector given by the joystick and delivers "control data" to the robot in order to make it move. Demonstration RTCs were made to run in the OpenRTM-aist platform, each one delivers and/or produces different data structures (Fig. 8), they are as following:

- Joystick RTC:

Delivers:<DIRECTION>([X_AXIS:1.0][Y_AXIS:1.0] [ROTATION:1.5]).

- Speed RTC:

Delivers:<DIRECTION>([SPEED: 2.0]).

- Multiplier:

Receives:<DIRECTION>([X_AXIS:1.0][Y_AXIS:1.0] [ROTATION:1.5][SPEED:2.0]).

Delivers:<CONTROL_DATA>([X_AXIS:2.0] [Y_AXIS:2.0][ROTATION:3.0]).

- RobotControl and RobotDisplay RTCs:

Receives:<CONTROL_DATA>([X_AXIS:2.0] [Y_AXIS:2.0] [ROTATION:3.0]).

Figure 7. Omnidirectional robot platform for testing.

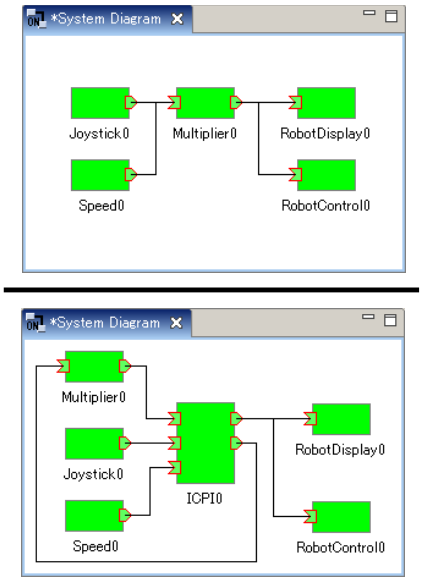

Figure 8. The described case study without (up) and with (down) the usage of the ICPI Client.

In this example, the data read and write cycle time while the AR module is performing background work triggered by the checking state is in average 150 ms, and when the system becomes stable (i.e. there are no RTCs to be categorized, related and managed) the average read and write cycle time is 10 ms. However, if the number of RTCs in the system increases, this cycle time also tends to increase.

TABLE 1. SPECIFICATIONS OF PC USED FOR TESTING

Processor	2.4GHz Intel Core i5-520M
RAM Memory	2 GB
Operating System	MS Windows 7 Pro. (32-bit)
Platform Version	OpenRTM-aist v1.0.0.0

V. CONCLUSION AND DISCUSSION

This article presented an introduction of the ICPI Client and Server along with a manager module prototype, the AR module, which identifies the roles of the software elements and/or applications and administrates them along with their related data, making possible the automatic building of a simple robot application and data complementation throughout an ICPI Client.

It was shown how an ICPI Client can help to interact with other interfaces/modules residing in an ICPI Server, the module tested here was the AR module, which makes easier in some extent the use of the OpenRTM-aist platform, since the user only needs to run the RTCs, define a speed value and use the joystick to move the omnidirectional platform in the desired direction.

In the example presented here, only how to organize, merge and redirect data between the RTCs is taken in account. Situation context, real-time response, and command interpretation issues will be considered in future works. They will also include the improvement of the ICPI with its modules by expanding their capabilities and features (e.g. the development of a GUI as shown in Fig. 9). Also testing of the ICPI with different clients and servers, as well as various hardware and functions in a variety of scenarios and challenges is to be performed.

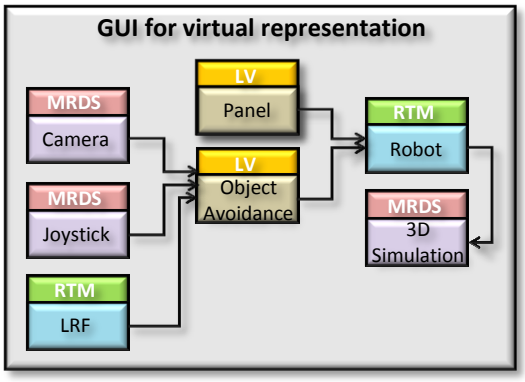

Figure 9. GUI example for an ICPI Server.

ACKNOWLEDGMENT

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Tokyo Institute of Technology.

REFERENCES

[1] Mizukawa, M.; "Robot technology (RT) trend and standardization", 2005 IEEE Workshop on Advanced Robotics and its Social Impacts, pp. 249- 253, 12-15 June 2005.

[2] Mohamed, N.; Al-Jaroodi, J. & Jawhar, I.; "Middleware for Robotics: A Survey", 2008 IEEE International Conference on Robotics, Automation, and Mechatronics, pp. 736-742, Sep. 2008.

[3] Quigley, M. et al.; "ROS: an open-source Robot Operating System", ICRA Workshop on Open Source Software, 2009.

[4] Robot Operating System, http://www.ros.org

[5] Fitzpatrick, P.; Metta, G. & Natale, L.; "Towards long-lived robot genes", Robotics and Autonomous Systems, vol. 56, pp. 29-45, 2008.

[6] Metta, G.; Fitzpatrick, P. & Natale, L.; "YARP: Yet Another Robot Platform", International Journal on Advanced Robotics Systems, vol. 3 (1), pp. 43-48, 2006.

[7] MS Robotics Developer Studio 4 ®, http://www.microsoft.com/robotics/

[8] Nielsen, H.; Chrysanthakopoulos, G.; "Decentralized Software Services Protocol – DSSP/1.0", Microsoft Open Specification Promise, 2006.

[9] NI LabVIEW ®, http://www.ni.com/labview/

[10] Ando, N.; Suehiro, T.; Kitagaki, K.; Kotoku, T. & Woo-Keun Yoon; "RT-middleware: distributed component middleware for RT (robot technology)", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.3933-3938, 2-6 Aug. 2005.

[11] Object Management Group, http://www.omg.org