Implementing Human-Robot Interaction Applications with GIMnet/MaCI

Seppo Heikkilä

***Abstract*—This paper presents how GIMnet/MaCI framework can be applied for human-robot interaction research. The example describes how WorkPartner service robot task communication application was implemented with GIMnet/MaCI by connecting together speech recognition, speech synthesis, manipulator control, and platform mobility MaCI modules. The implemented system was used as such both with simulated and real WorkPartner robot, demonstrating the GIMnet/MaCI flexibility to control different robots with one implemented software. The paper shows in a concrete implementation level detail how it is to implement HRI application with GIMnet/MaCI framework and further use it in a HRI research.**

I. Introduction

This paper describes experiences of using GIM/MaCI framework [3] for a human-robot interaction (HRI) research. The idea of the paper is to provide detailed description of how the HRI software development can be done with GIM/MaCI framework and thus help people considering to use the system for similar purposes. The examined example is the software architecture developed for the SpacePartner project astronaut-robot cooperation research.

SpacePartner project is a co-sponsored PhD project of the European Space Agency (ESA) and the Aalto University. The project was initiated under the ESA Network Partnering Initiative (NPI) program, which target is to increase interaction between the ESA and European universities. The idea of ESA NPI program is also to enhance space research through spin-ins from advanced non-space projects. In this case, the spin-in is to utilize the existing WorkPartner service robot to develop astronaut-robot cooperative task definition and execution capabilities.

The WorkPartner service robot, or the future SpacePartner, in shown in Figure 1. The WorkPartner robot and the SpacePartner project has been introduced with more details in [1], [2].

Figure 1. WorkPartner service robot working as an astronaut assistance.

II. Developed System Structure

The purpose of the developed software is to enable astronaut and robot communicate two different tasks. The first task is doing measurements of wanted sample and the second task is inserting a measurement unit to pointed location on the ground. Basically the user has to thus communicate action and target.

The structure of the developed software is shown in Figure 2. The software is basically distributed to three different computers. First one is the computer onboard the robot, which creates MaCI modules interfaces to the robot devices, e.g. SICK laser scanner. The second computer is a server computer which is running all the computation intensive software modules, such as laser scan based odometry calculations. The third computer is carried by the user and provides MaCI modules that enable to receive the user communications.

All the arrows between modules that have not been named in the Figure are using GIM/MaCI communication and are connect through GIM/MaCI tcpHub [3] which a core server of the GIMnet network. TcpHub enables modules to announce the services that they can provide and share data to multiple clients. For example when the laser scanner data in this example is send to tcpHub it is distributed directly to bbSLAM and human localisation clients.

III. Creating An User Interface Service

In order to get a concrete idea how an user interface module can be added to the above software, lets examine how the user localisation module was constructed. The idea of this module is to locate the user from the robot laser scanner data. This means that the input of the module is laser scanner data, i.e. input from MaCI Ranging server, and output of the module is the location of the user. Thus the module is a service that is outputing position data, i.e. MaCI position server.

The basic idea with GIM/MaCI framework is that there is an example implementation of server and client for each type of module. In our case the module type is Position module as we provide Position interface. There is already implementation of Position module which sends random location data and an example of client that can read the position data. We can use the same client of course here and we can take the dummy Position module example as our starting point.

All types of MaCI modules have similar GIMnet initialisation code, shown in Algorithm 1. The GIMI class provides the basic communication capabilites for the module and takes care of the connection to the tcpHub which enables servers to sent their data and clients to receive it. Second part is

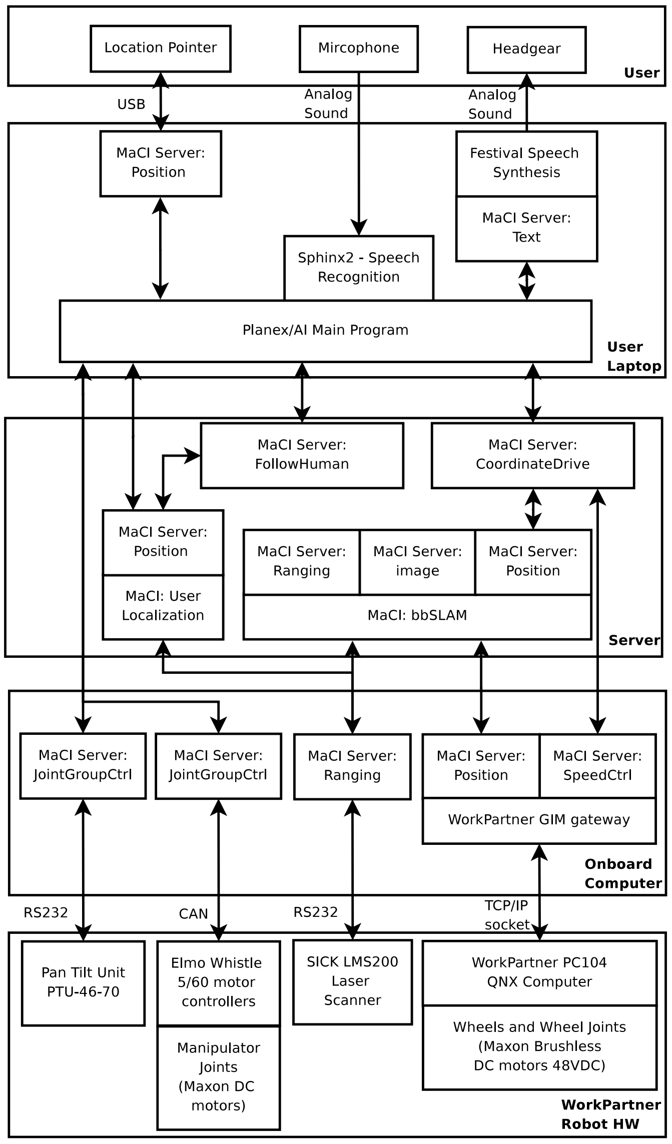


Figure 2. GIM/MaCI WorkPartner Human-Robot Interaction(HRI) software structure. All arrows without description are communication through tcpHub and each MaCI server has a MaCI client in the other end of the arrow.

Algorithm 1 GIMI initialisation algorithm connecting to tcpHub. Example values for the variables: tcpHub_IP = ”as-robo.hut.fi”, tcpHub_Port = 50002, gimnetName = ”Default” and maciGroupName = ”WorkPartner”.

```
#include "gimi.h"
// Create GIMI class and connect to tcpHub
gimi::GIMI g;
g.connectToHubEx(tcpHub_IP, tcpHub_Port, gimnetName);
// Create MaCICtrl instance and set group name
MaCICtrl::CMaCICtrlServer mcs(&g);
mcs.SetGroupName(maciGroupName);
```

creating of a MaCI instance, which provides generic Machine Control Interface (MaCI) services, such as naming system. The most important MaCI property is probably to set the MaCI name which is basically name of the system where the service belongs, e.g. WorkPartner robot in our case.

In this point we have established connection to the tcpHub.

Algorithm 2 Creating Position server and establishing it to the tcpHub. Example values for the variables: interfaceInstanceName = “Pose”.

```
#include "PositionServer.hpp"
// Create position server class, name it and open it
Position::CPositionServer
ps(&mcs, -1);
ps.SetInterfaceInstanceName(interfaceInstanceName);
ps.Open();
```

Algorithm 3 Sending position data to the clients. Example values for the variables: x=y=a=vx=vy=va=0, p=0.9.

```
// Create position data and send it to the server
CPositionData pos;
pos.CreateInternalBinBag();
pos.SetPose2D(TPose2D(x,y,a));
pos.SetVariance2D(TVariance2D(vx,vy,va));
pos.SetProbability(TProbability(p));
ps.SendPositionEvent(pos);
```

Next we need to create the position server. This is shown in Algorithm 2. First, we need to connect the Position module to the MaCICtrl class we created earlier and set the name for the interface, e.g. Pose. After that we need to only open the service in order it to appear to the tcpHub.

Naturally, it is not enough to only have the service established on the tcpHub but we need also to handle the data going and coming from the server. In Position interface case especially important is of course the sending of the data. The code shown in Algorithm 3 describes how data is send to the tcpHub. The idea of the InternalBinBag is to encode the data for sending. In real case you might want to run this code in a loop on a separate thread and you would need to implement also the algorithm that calculates the human location.

This is shortly how a Position service is set up in practise. The Position server is push type of interface, i.e. the data is not polled by the clients but pushed by the service to the clients. Most services in GIM/MaCI which are providing data are implemented in this push type of way.

IV. Creating Client Module For Server

Creation of a client starts exactly like creation of a service, i.e. with the code shown in Algorithm 1. After the connection is established to the tcpHub, the process of opening of service is a bit different as the service is searched using so called Service Locator (SL) name. Code showing this is shown in Algorithm 4.

Next we need to just keep checking if there is new position data available. This can be done with the code shown in

Algorithm 4 Creating a client that connects to the Position server. Example values for the variables: MaCISL_name = “WorkPartner.MaCI_Position.Pose”, timeout_ms = 10000.

```
#include "PositionClient.hpp"
// Create position client class
MaCI::Position::CPositionClient pc(&g, 0);
MaCI::MaCICtrl::SMaCISL sl(MaCISL_name);
pc.SetDefaultTarget(sl, timeout_ms);
```

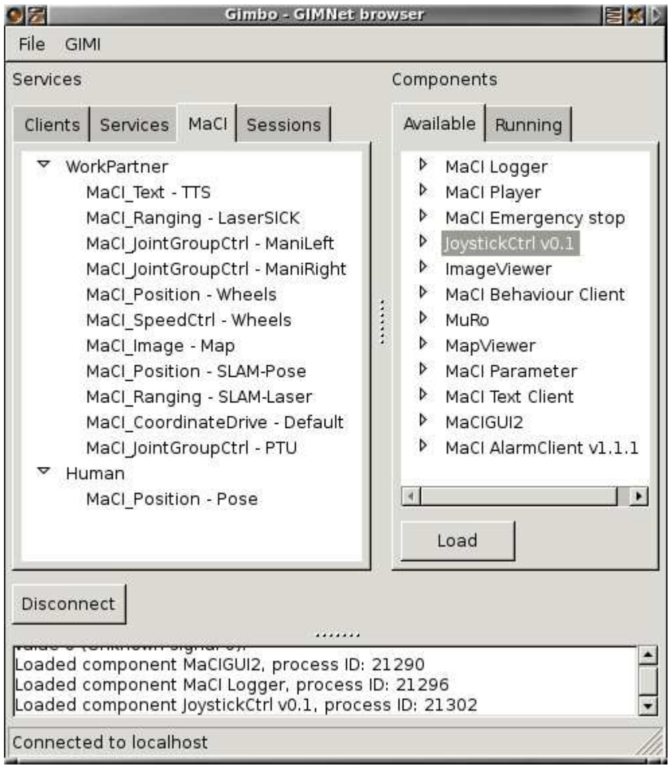
Algorithm 5. The sequence parameter makes sure that all the available data will be read one after the other.

Algorithm 5 Example code of Position client data receiving. Example values for the variables: timeout_ms = 1000. A non NULL sequence parameter is required in case all the data needs to be read and processed.

```
// Check if new data is available and print it
CPositionData posData;
int sequence = -1;
bool dataAvailable = pc.GetPositionEvent(posData, &sequence
     , timeout);
if(dataAvailable)
  posData.Print(1);
```

V. tcpHub View

When the system shown in Figure 2 is fully started up, we get several services initialized to our tcpHub. The Figure 3 shows how these services look on a tcpHub with a program called Gimbo. Gimbo enables for example to browse available MaCIgroups, i.e. basically machines, and the services they provide, e.g. Position.



```
Loaded component MaCIGUI2, process ID: 21290
Loaded component MaCI Logger, process ID: 21296
Loaded component JoystickCtrl v0.1, process ID: 21302
```

Figure 3. Gimbo service view with all WorkPartner services on the left and all the Gimbo clients on the right.

The Gimbo can launch separate programs, suchs as JoystickCtrl, which are generic clients that can use the provided services. The reason for launching separate programs is that if one of the programs crashes it does not affect Gimbo or other programs.

Some of these separate programs in Gimbo, that you might want to use, are Logger, saves data sent by selected modules, Player, reads and “plays” previously saved data from filesystem, and JoystickCtrl, enables to control JointGroupCtrl services and SpeedCtrl services. In addition MuRo and MaCIGUI2 are useful to visualize data, e.g. from position and ranging services.

VI. Discussion

The implemented system has been tested with user tests and it proved that GIM/MaCI framework was able to work as a backbone of a HRI research software.

Top five advantages of GIM/MaCI framework

- Amount of pre-existing modules, such as drivers and algorithms with implemented "ready to use" server and client modules.
- Speed of communication, around 1ms back and worth response times can be expected in LAN.
- Stability, the core functionalities have never failed.
- System distributability, only driver modules need to be running on the robot or in the user laptop.
- Modularity, the code is forced to be inherently modular with standard MaCI interfaces which makes it more easily manageable.

Top five weak points of GIM/MaCI framework

- Lack of implementation examples. For example, how a complete robot could be implemented to be GIM/MaCI compatible.
- Lack of theory documentation for MaCI, in addition to SI units. For example, how angles and coordinates are defined.
- Number of pre-existing user interface modules, basically only GUIs are available at the moment.
- Establishing communication takes time. Service Locator (SL) search, to find the available services, can take up to ten seconds.
- Modules have sometimes problems to connect to tcpHub in high-delay wireless networks.

VII. Conclusions

The GIM/MaCI framework implementation for HRI research has been described. The modular and distributable structure of the GIM/MaCI framework was shown to have the flexibility required by inherently distributed HRI applications. Although the user interface development has not been core target of the GIM/MaCI framework, the mature core capabilities and existing examples makes user interface development a straight forward process.

References

[1] Seppo Heikkilä. Spacepartner - robotic astronaut assistant. In Kalevi Huhtala and Janne Uusi-Heikkilä, editors, *Proceedings of the second workshop on generic intelligent machines*, Tampere, 2009. Tampereen teknillinen yliopisto.

[2] Seppo Heikkilä, Frederic Didot, and Aarne Halme. Centaur-type service robot technology assessment for astronaut assistant development. In *Proc. 10th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA)*, Noordwijk, Netherlands, November 2008.

[3] Jari Saarinen, Antti Maula, Renne Nissinen, Harri Kukkonen, Jussi Suomela, and Aarne Halme. Gimnet - infrastructure for distributed control of generic intelligent machines. In *Proceedings of the 13th IASTED International Conference on Robotics and Applications Telematics 2007*, 2007. The 13th IASTED International Conference on Robotics and Applications Telematics.