Interface for Controlling a Fleet of Generic Machines

Matthieu Myrsky ∗ Antti Maula ∗ Jari Saarinen ∗

∗ Aalto University, School of Science and Technology Department of Automation and Systems Technology, Finland (e-mail: firstname.lastname@aalto.fi).

Abstract: This paper presents a dynamic interface for controlling a machine and a fleet of machines. A distributed communication framework, named GIMnet, provides in addition to service based communication, a tool listing all the available services in the network. On top of GIMnet, a hardware abstraction layer, named MaCI, is developed which is used to control and provide information from robots in a generic way. All the services that robots are providing are named so, that they can be identified from which robot and which device they belong to. Based on the service discovery, naming of services and the clients of MaCI, an interface for connecting and controlling a machine or a group of machines is presented. In case of new services from the machine appears, the interface dynamically connects to them and so provides a way to get information from it. In this work, as a use case, the interface is used to control a group of generic machines in a wanted formation.

Keywords: robot controlling software, fleet management, multimachine communication, multimachine interface,

1. INTRODUCTION

Mobile robots are becoming more common and their abilities more elaborate in research as well as in the industry. This increases the number of tasks where robots can be used and it has become the reality that a worksite has multiple machines that are fully or partially automated. Work environments where groups of autonomous machines are used are harbours, mines, warehouses etc. This raises the need of easily get access to a group of machines and give commands to them.

The widely used robot controlling middleware are usually distributed. They abstract the hardware and robot-related information at some level. The tasks are becoming more complex which causes the software components are combining more and more information sources to achieve the goal. Also the usage of group of machines can be necessary to achieve greater goals. This in mind, the usage of an interface allowing to control a whole machine or a group of machines with wanted abilities is reasonable.

In this paper, an interface for controlling a generic fleet of machines is described. The interface uses a communication infrastructure named GIMnet and a hardware abstraction layer named MaCI which uses GIMnet for communication. GIMnet is a service-based communication middleware for distributed applications. It provides a tool for scanning the network and list all the services found on that network. MaCI provides specific information about these services such as what kind of information is available and how they are related to the machine.

With this interface controlling a generic fleet of machines is shown to be possible and it also speeds up the development process. The benefit of a generic interface is to have not to be aware in advance of the machines nor their number. The abilities of the machines can be found out dynamically and take under control the machines with necessary services for the wanted task.

To demonstrate the interface, a simple teleoperation of a fleet of machines is presented. A fleet controlling application controls a fleet of machines found in the network. One of the machines is defined as a master and the slaves are put to a wanted position. By controlling the master, the application controls the slaves to maintain the formation.

This paper is divided into five section, starting with the introduction. The second part is presenting the related work related to the topic. Third section is the system description including the communication infrastructure, the hardware abstraction layer and the fleet controlling interface. The fourth section presents a multi-machine teleoperation use case, where the proposed interface is used. In the fifth section, we conclude the paper and discuss the future work.

2. RELATED WORK

Robots can be considered as a group of sensors and actuators which are controlled by a computer. The information from the sensors is processed to have an understanding of the environment. Based on this information, the robot is controlled. To get all this work done, multiple robotic middlewares has been implemented to simplify this process. In the work of Mohamed et al. (2008), a survey of different robotics middleware are discussed.

Most of the robotic middlewares are primarily made for controlling one robot. RT-middleware (Ando et al. (2005)) is using CORBA as communication middleware and the
main goal was to simplify the process of building robots with a modular software structure. Another CORBA based middleware is Orca (Malarenko et al. (2006)) with the main goal in software reuse in robotics applications. Other well-known middlewares are MARIE (Cote et al. (2006)), Miro (Enderle et al. (2001), Player and ROS. These last two seem to be the most De facto robotic middlewares and they are introduced below. In this related work section we concentrate how some of the most used middlewares and middlewares specially developed from the perspective of multi-robot issues are dealing with multi-robot communication and controlling.

Player (Collett et al. (2005)) is a well-known and widely used robot controlling middleware and has been used to research multi-robot systems (Gerkey et al. (2003)). To control multiple robots there are two solutions. Either connecting each robot with separate clients to predefined ports or to create a “passthrough”-player server. It simply connects to the wanted and known robots and is seen as one group of devices in one TCP port. This port can then be connected with one player client.

ROS (Quigley et al. (2009)), Robot Operating System, is probably the leading middleware. It provides libraries and tools for creating robot applications. It has a wide community providing new features for it. ROS uses direct communication between nodes. It has a Master core which acts as a nameservice in the system. Nodes register to the master and have information about other nodes on the network. For multi-robot communication, the nodes must connect other robot nodes directly. This can be a problem if robots are communication behind a slow link and many nodes are interested in its data. This can be solved by a Proxy Ros-stack which acts as a repeater node. As in Player, multi-robot communication requires additional software components. Also none dedicated interfaces were found when researching the multi-robot communication in these systems.

CoRoBa, a multi mobile robot control framework uses CORBA as communication middleware ( Colon et al. (2006)). In this system, components can be added, discovered, located at run-time by using the name service provided by CORBA. Although, the framework is called a multi mobile robot control framework, it doesn’t support any specialized tool for connecting a fleet of robots. As a future work they mentioned a general distributed behaviour engine for multi-robot application, but any additional information of it was not found.

In the work of Long et al. (2005), a Distributed Field Robot Architecture (DFRA) is presented. It uses Jini as network architecture for constructing distributed systems. Jini offers lots of tools, like the service discovery, which is used in this work to find robots in the network. A generic waypoint-following behavior is constructed by using DFRA. At the activation of the behavior, it requests and waits for the predefined group of interfaces of the same robot. Based on the documentation found of this architecture, it doesn’t offer any generic tool for controlling robot fleets.

ROCI (Chaimowicz et al. (2003)), is a distributed framework for multi-robot perception and control. It is divided into modules, tasks and nodes. Modules are small computational blocks which have an input, processes it and the provides something as an output. Tasks are a group of modules that are connected together to perform a specific task and nodes are representing a robot. This work presents a multi-robot scenario where a group of robots are taking images of the environment and combine it to the actual position got from the robot’s GPS device. In a case where the signal of the GPS device is lost, a new module for localization is created and dynamically launched. The module searches all other nodes which are seen in its camera and use their GPS position as input. With these information sources, it calculates an estimate of its position and this information can be added to the taken picture.

These two previous examples have the same approach to control a fleet of machines than in our work. The main difference is that in our approach, we have done a generic interface to connect a fleet instead of using different tools to build an application which controls a fleet.

3. SYSTEM DESCRIPTION

3.1 GIMnet

GIMnet (Saarinen et al. (2007) and Maula et al. (2012)) is a communication infrastructure designed for robotics application. It is a service based communication middleware for building distributed applications. As the main features, it provides distributed publish/subscribe mechanism, dynamic discovery of services and runtime service binding.

Figure 1 show the principle of GIMnet. GIMnetAP is the network layer of GIMnet and handles transferring packets between application nodes. GIMnet can consist of many APs which are forming a Virtual Private Network (VPN). Application nodes connect to GIMnetAP and can communicate with any of the nodes connected to the same network.

GIMnet has two core services which are not (directly) seen by the user. Nameservice provides a name to address and vice versa. Application nodes are known in GIMnet by names, so applications don’t have to be aware of the physical location of the service. Multicast service is used to duplicate sent data to all. It is used by GIMnet operations such as GIMnet service discovery.

GIMI is the GIMnet-interface for application nodes. Its API gives the tools to point-to-point messaging between nodes, subscription based messaging and means to locate and utilize services dynamically from the network. GIMI is
implemented as a C++ API and is working on Linux operating systems. GIMI is distributed as open source and can be downloaded from its web page (gim.tkk.fi/GIMnet).

Together the GIMI, GIMnetAP, APServices and the application nodes form the communication framework known as GIMnet.

**Service discovery** GIMnet provides a dynamic and fast service discovery of the network. It lists all active services of the network. This enables connecting machines or services which are not defined in advance.

Service discovery is based on the Multicast APService, which is used to distribute the service query to services. When an application node connects to GIMnet, it will register itself as a subscriber for service discovery queries. This allows the Multicast service to know exactly how many clients are active in the network. The service discovery itself is executed by sending a single probe message to the multicast service. The multicast service distributes the query to subscribers. Final step of the service discovery is to wait until the service providers respond directly to the calling application node. The service discovery is a really fast procedure. It is shown in Maula et al. (2012) that it takes for example only 70 ms to find out 1000 services which are in the network.

### 3.2 MaCI

MaCI is a component-based library for building robot controlling software, which uses GIMnet as a backend for communication. It is acting as a hardware abstraction layer (HAL) by strictly defining interfaces for communication. It acts mostly as the same role as other robot controlling software mentioned in the Related Work-section.

As MaCI uses GIMnet, it is also implemented with C++ and running under Linux operating systems. Some of the MaCI-interfaces (described below) are wrapped in Python which gives the possibility to develop applications also with Python. The wrapping is done by using a tool named SWIG (Beazley et al. (1996)). MaCI is also published as open source and can be downloaded from its web page (gim.tkk.fi/MaCI).

**MaCI-interface** Interfaces in MaCI are for communicating over GIMnet with a standard way. Every interface defines the datastructure that is transferred using GIMnet, and provides client and server side API. GIMnet is hidden from the MaCI-user, such as the user doesn’t use GIMI-function to send data but more functional calls such as set speed, send position data etc. instead of send to subscribers-function which is used at GIMI-layer.

The structure of every interface is similar and they are split in three different parts: Datatypes, data interpreter and to a client and a server (Figure 2).

Datatypes describe the atoms inside the communication packet. It defines the types (int, float, char etc) of all data of this interface moving. It also encapsulates datatypes in structures to define which values are to be represented together.

![Fig. 2. Idea of MaCI-interface](image)

Data interpreter class is for packing datatypes in a container. It handles the adding and getting of all datatypes used in this interface.

Client and servers are for sending and receiving commands or data with data interpreter class. Clients and servers usually uses and handles the data interpreter class and provides the easiest possible way to the user to use this class.

The client-server model differs from a basic client-server architecture, where the client requests a service from the server and wait for the response (Tanenbaum and Steen (2002).) In MaCI, the server provides information or receives commands and clients’ main feature is to get provided information or to give commands to the connected server. Data flow is aimed to be event-based so the receiver only waits for the newest data from the sender instead of asking for new information periodically.

At the moment there are about 20 different interfaces describing different kind of information. As example there are interfaces for the position, maps, images, speed controlling, ranging devices, IMU, driving to coordinates, energy etc.

**MaCI-module** Modules are standalone processes which are providing a service. There are three different types of modules which can be seen in Figure 3. In the simplest case, a module uses one device and provides one service. It is also possible that a module provides multiple services. For example, a motor controller, in addition to command the wheel to rotate, provides position information calculated from the rotation of the wheel. The third module type is when a module binds a client(s) and a server(s), as is the case on top where a localisation module takes for an input an odometry and a ranging information and provides more accurate position information. A robot is usually a group of modules, that are connected to each other.

**MaCICtrl** MaCI has its own naming policy which differs from the naming policy of GIMnet and uses MaCICtrl-interface which is a utility interface. MaCICtrl has a naming system called MaCISL. Instead of a freeform name of GIMnet, the policy of MaCISL naming is divided in three parts, group, interface and instance names. MaCISL name is the sum of these three and all these parts are separated with dots. MaCICtrl-utility interface provides functions to easily get appropriate information such as...
MachineCtrl and FleetCtrl-interfaces are basically interfaces such as other interfaces. It provides the developer an easy access to a whole machine or to a fleet of machines. As tools, it uses the GIMnet service discovery, the MaCICtrl-utility interface and the clients of all MaCI-interfaces.

The user gives the hostname and port of the GIMnet access point and the MaCI group name describing the machine. On initialisation, the client scans the network and creates appropriate clients for all the MaCI-interfaces found which belongs to the given group name.

After initialisation, the user has a group of client ready to connect the services. Depending on the needs of the user, it can connect all the clients or just the wanted clients. To avoid unnecessary communication traffic the user should open only the clients that is will be used.

The interface also has tools to find out, what kind of interfaces the machine has and how many instances of the specific interface are available. Also functions to get the information about services, as the list of names, are available.

MachineCtrl-interface provides “Getters” to the pointers of clients that have been created. A specific client can be obtained by giving the instance name of it or a c++-map of a specific interface. The map contains the instance name and the pointer to the client.

The fleetCtrl-interface is an extension to the machine interface. On the initialization phase, instead of giving the exact group name, a part of the group name can be give. As MaCI-group name can be extended so long as wanted, by giving e.g. the group name up to the fleet level, the fleetctrl-interface then initializes all the machines under this fleet. As a simplified example, GIMnet has the following services with MaCICtrl-names:

- Worksite.Digger.MaCI_Position.SLAM
- Worksite.Digger.MaCI_Ranging.laserScanner
- Worksite.Loader.MaCI_Position.SLAM
- Worksite.Loader.MaCI_Ranging.laserScanner

The worksite has two machines, Digger and Loader, which both of them have two services. One providing SLAM position and one laserScanner data. By giving the FleetCtrl-interface 'Worksite' as a fleet name, it will find two machines (Digger and Loader) from the network. It initialises two machines which both initialize one Position client and one Ranging client by using machineCtrl-interface.

The Fleet interface can, if the user wants, scan the network continuously to check for new machines connecting the network. If a new machine appears, it automatically creates appropriate clients ready to connect the machines. This feature is naturally available for the machineCtrl-interface.

4. USE CASE - MULTI-MACHINE TELEOPERATION

As a use case of usage of FleetCtrl-interface, we made a multi-machine teleoperation test. The number or the type of the machines is undefined in advance.
Our application connects to GIMnet and connects to the wanted machines or to a complete fleet of machines. It has a GUI where the machines can be arranged to a wanted formation. The requirement for accepting the robot to the application is that it is providing some kind of position information and a speed control interface to be able to control the robot. One of the machines is selected as a master machine and the others are slaves. The application moves the slaves when the master is moving trying to keep their position the same relative to the master.

As the formation can contain machines of different kind, which have different abilities, it has to limit its maximum speed to the speed of the slowest machine in the fleet. For this reason, the application creates a new speedCtrl service with this limitation and appears at the network instantly under the same group name as the machine chosen as a master.

To keep the slaves in the correct position, the application uses as information the speed given by the teleoperator, the position of the master and the position of the slave machine. The master is controlled directly with the speed given by the teleoperator. The slave speed depends on their relative position to the master that they should keep. They reduce their speed if they are too near or accelerate in case they are too far away. The communication diagram of the application is described in Figure 4. The control loop for every machine is updated when slaves send new position information.

The teleoperation was done with a separate GUI which shows the position of each machine and enables the robot to be controlled with the arrow keys of the keyboard.

We have tested this application with two different use cases. In the development phase, a simulator was used. It has four different machine types, three different sized differential wheeled robot and warehouse AGV. All of them provides global position information and a speed controlling interface (by speed and angular speed). The machines have different maximum moving speeds and kinematics. Also they are sending position data with a different interval from 10 Hz to 20 Hz. During the development and testing phase the simulator was run in the same machine as well as in a different machine connected in a 100 Mb network.

In a real world demonstration, we used six differential wheeled robots of the same type (see Figure 6) providing odometry information and ranging information from laser scanners. In addition a map of the surrounding was available in the network, which was generated from laser-scanner data collected by driving one robot in the area. The robots were localizing themselves by using odometry, laserscanner and map data and providing the processed position information to the network. The localization data was provided to the network by each robot every 50 ms which makes a frequency of 20 Hz. As the controller calculates new speed commands to the slaves each time that a new position is arrived, it makes that the control is executed at 100 Hz in this experiment. The machines were communication through WLAN and the controlling was done on a desktop computer.

As results, the FleetCtrl-interface found all the machines in the network and connect them with no problem during both of these tests. Also, the controller was able to control the machines and keep them in the wanted formation in both of the experiments. The controller was running on a normal desktop computer and could control without
running out of bandwidth or processing time. Using the simulator, the application using the interface, providing the GUI and controlling the machines, took 12% of the processing power of a 2.4GHz processor when controlling 9 machines with a control loop running at 150 Hz. Data rate of the received data from simulator to the application was in average 64 kb/s and of the sent data 53 kb/s.

It is clear that when the number of machines is growing, depending on the network or the computer used either of these is running out at some point. However, the used interface doesn’t produce significant additional network traffic, delays or computational cost compared to use the MaCI-interfaces separately. The only addition is the data rate produced by making the service discovery. This can be easily reduced or completely removed by changing the update interval or disabling automatic update.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a dynamic interface for controlling generic machine fleets. We demonstrated it by teleoperating a fleet of machines. The interface simplifies building applications for fleet of machines. All of the services are constructed and ready to use simply by connecting to GIMnet and giving the name of the fleet.

The interface enables the development of multi-machine applications without knowing the number of machines in advance. Also the hardware abstraction layer enables the development without knowing the exact configuration or device types. The machines can be filtered on the basis of services which they are providing and deduce their capabilities based on their services. In addition if new services or machines are connected to the system, new clients are automatically created.

Although it basically only creates clients and provides functions to connect some or all of them. Only information about a specific client can be achieved from the name of the instance. Also MaCI-interfaces can provide some specific data from the instance, for example the position of a device. However the main missing feature is how the different modules are connected to each other. So it is impossible to deduce automatically, in a case where multiple instances of the same interface are available, which instance is giving the most accurate information.

Another missing feature is a mechanism to inform the user of the interface when new services or new robots connect to the network. The current way if is to poll the number of services of a machine or the number of machines. To simplify it could be easily done by implementing a callback function which is called by the interface when new services are available.

ACKNOWLEDGEMENTS

This work has been supported by the Academy of Sciences in context of Centre of Excellence in Generic Intelligent Machines, and by Tekes/FIMECC project MOTTI. The support is gratefully acknowledged.

REFERENCES


