ICE Extension of RT-Middleware Framework

Zoltán Krizsán

University of Miskolc, Hungary
krizsan@iit.uni-miskolc.hu

Abstract: There is a promising middleware framework for robot systems called RT-Middleware (and its open source implementation named OpenRTM-aist). A rising demand exists for extending the existing OpenRTM-aist middleware with the fast communication capability of ICE overcoming the communication speed issues of the original version. With the help of ICE extension the communication among components has been direct and fast, on the other hand, the modularization, and creation of systems stay simple and easy. Another important improvementof the suggested extension is the embadded possibility for creating a connection among one consumer to multiple providers. The installation of ICE extension is a simple file copy operation, so the base system can still work futher with no changes in the original configuration. Upgrading the existing components to the the suggested new extension is simple by replacing the CorbaPort type to IcePort. The main contribution of the paper is the detailed introduction of the suggested changes for the OpenRTM-aist middleware. The paper also discuss the concept of this extension.

Keywords: robotics, middleware, distributed systems, object oriented programming, OpenRTM-aist

1 Introduction

The development of robot systems uses different robot parts (sensors, processing elements, actuators, data trasformer) which are combined with each other in a compact, self contained system. The quick and easy development, the system reconfigurability (at runtime, without recompiling) and the flexibility demand the introduction of middleware frameworks for robot systems. One promising middleware framework is the RT-Middleware technology (and its open source implementation named OpenRTM-aist). For solving the speed issues of the original system, a new Internet Communication Engine (ICE) approach was introduced. Using ICE for Remote Method Invocation (RMI) in robot systems is more efficient than other currently applyied RMIs. There was a rising demand for extending the existing OpenRTM-aist middleware with the fast communication capability of ICE. Building robot systems contain more parts, is frequent process in the researcher life and the industrial area too. The environment of these systems can change often and the requirements are also changeable, so the need for a rebuilding and reconfiguration tool is essential. A common protocol and the modularized concept are key issues solving this problem. The middleware technology was introduced for this express purpose. There is a robot middleware standard which can be found among Object Management Group (OMG).

1.1 The RT-Middleware Technology

The specification of RT-Middleware is defined in “Robotic Technology Component 1.0” and “Super Distributed Object 1.1” which can be found in [1]. These specifications describe the concept of the structure of modularized robot system and the system behavior. In the RT-Middleware, the software modularized into components of RT functional element is called **RT-Component (RTC)**. Each RT-Component has the interface called Port, to communicate with other components or exchange data. The RT system is constructed by connecting the ports of multiple components as an aggregation of each RT-Component function. The OpenRTM-aist is an open source implementation (it can be downloaded from [3]) of the RT-Middleware specification based on Corba technology. It is developed by National Institute of Advanced Industrial Science and Technology - Intelligent Systems Research Institute - Task Intelligence Research Group.

1.2 The ICE Technology

The ICE Framework is an object oriented distribution platform that could run on several different operating systems and implementation languages. It represents a new approach to middleware that builds on Corba’s strengths while avoiding its weakness. The Corba was introduced in the early nineties which fulfill the requirements of that ages. Nowadays the Corba is out of date, it has got many limitations, and can be difficult to learn and complex to use.

The most important part of ICE is the upgraded object model. Using a remote object does not require additional service such as naming or bootstrap service. It uses Proxies which are handles the remote object in transparent way. As a result, applications require less code and the finished system has fewer dependencies on external services that might fail.

ICE provides not only interface inheritance but also interface aggregation. A client can demand an interface from a server. If the object supports the requested interface, the ICE runtime creates a new proxy for that. The interface aggregation solves the problem of versioning, because the developers can add newer interfaces to the existing objects without violating the client-server contract.

ICE uses new specification language for describing the remote object interface, which is clear, simple and efficient. This uses minimal number of built-in primitive types, and it allows the usage of user-defined types and it has built-in dictionary which is a key-value pair storage structure.

The ICE is inherently multithreaded. On the server side, a thread pool using a leader-followed model dispatches incoming invocations. The clients also use thread pool for invocation. The system runtime itself is fully thread-safe and gives numerous items for developing multithreaded application.

The ICE protocol can run over whatever streams and datagram transports. Currently, ICE supports TCP/IP and ssl as stream transport, and UDP as datagram transport. The protocol engine is extensible, so developers can add new transport.

2 The RT-Middleware in Details

The advantage of OpenRTM-aist is the easy and user friendly way to use robot parts. There are two groups of users:

- Component developers: They know the programming languages and want to develop components in an easy way.
- End users (researcher): They would like to use only the system (prebuilt components) - without any programming knowledge – via a user friendly interface.

The middleware also contains a graphical editor, named RTC Builder, for component creation. After the component definition is created (stored in xml file) the system generates the appropriate source files. The work of developer is just filling the body of generated source code. A lots of applications have made using this system (f.e.: [4],[5]) which proves the easy and user friendly application.

After the new robot system is ready and the components running in different places, they can be explored by a graphical application called RTC system editor. The user can make a new system which is built from components and can activate it. The user can change the configuration by rtc.conf file before starting the components or real time via the graphical application. After the user closes the manager application the new system will run over.

Building blocks of the system are RT components (Robot) running separately which has well defined architecture. The acrhitecture and interfaces are shown in

Figure 1. Their connections (communications) are possible only via ports. The components can send/receive data to/from each other via dataport. The components live, work and die. It has more states that are controlled by the system and the user.

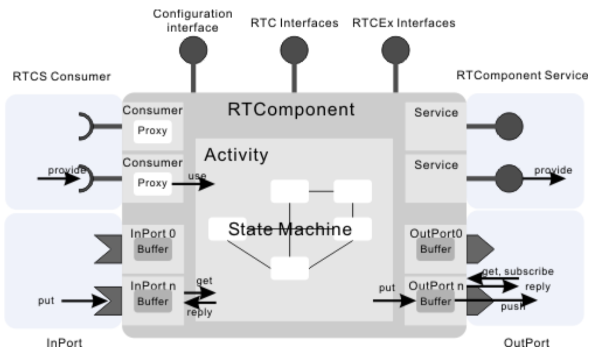

Figure 1
The architecture of RT Component.

It can have more ports: input ports and output ports (input: „get something„ from the other, output: “give something” to the other).

Two kinds of ports are exist:

- Data port: buffered stream, generally it has got a type.
    - Inport: This is the input port that handles the data received from other components. As the core logic will process data received by the InPorts sequentially, InPort is particularly well suited to periodic components.
    - Outport: This is the output port that will stream the processed data to other components. It supports both the pull mode, in which the receiver acquires the data by itself, and the push mode, in which the data is actively sent receivers that subscribed.
- Service port: This implements the Remote Method Invocation. The provider exposes its functionalities and data as remote object. The consumer can use the remote functionalities as local ones. The RTM system is based on CORBA middleware and it support CORBA service port. The provider implements (a servant object) the idl interface(s) (interface definition language).

2 The Suggested ICE Port Extension

The main goal of the suggested extension is to add fast ICE communication possibility to the components in form of IcePort. The new extension is implemented in dynamic library (dll on Windows, so in linux), so the installation is a simple file copy operation, and does not modify the base system functionalities just improves it. Tha main concept is: replacing the corba port class with the new IcePort, after the user makes the connections and ativates the components, the remote method invocation is made directly among the compoents without any Corba involvment. The parts of new improved system and their task are shown in the Figure 2.

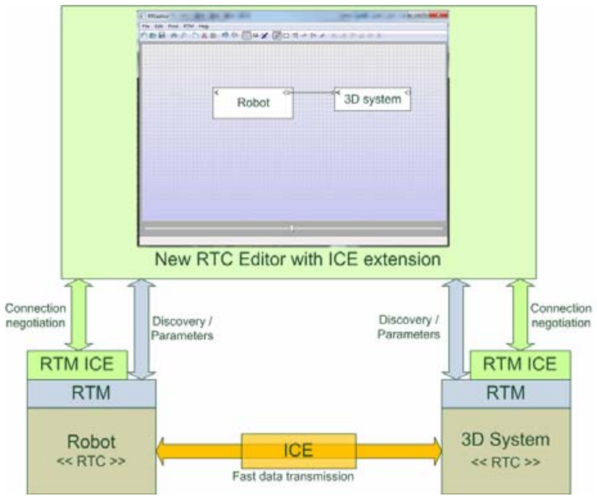

Figure 2
The main concept of ICE extension.

One component can have one or more Ice port(s), which is(are) service port(s). From this time forth there are two kind of service port: CorbaPort and IcePort. The IcePort can provide or consume one or more services (Ice interfaces). The slice language describes the functionalities of these interfaces, and the servant class implements that interface. If an ICE port acts as a provider it opens one TCP port for one ICE object adapter, so if the module has more IcePorts it opens more TCP ports too. If an Ice Port is a provider its name must be unique. If a component exists with such name the system throws an exception and the component will not be created.

The accustomed configuration file (rtc.conf) can contain definition for ice port number which can define the system port number for every IcePort in the following form:

ice_port.iceportname: number

If it does not contain the specified ice port number then it is assigned and increased automatically. If there is no definition for IcePort then the first ice port will be 20000 the second will be 20001, …The name of created ICE endpoint will be the name of IcePort. If the port is already in use (e.g. an other application uses the automatically assigned or required system port) then an exception is thrown and the component does not created.

```
Oct 16 17:15:05 ERROR: : the port 12382 is already used, please change or add "ice_port.server1:xxxx" entry with a free port number!!!
```

The IcePort supports the 1:N and N:1 connections. One consumer can use more providers and more consumers can use one provider. In case of one consumer uses more providers the developer can get the count of providers, and the method invocation is possible for all remote objects. The ice adapter will be activated after the port gets connected to other(s).

Because the original RTC system editor does not handle the IcePort, we had to write a new one. The new editor written in C++ and using WxWidget, which garantees the fast run and small size.

3 An Application Example: VIRCA

The Virtual Collaborating Arena (VIRCA) is making a virtual laboratory that gives a way to the team of researchers or industrial engineers to collaborate with each other control a physical device remotely in easy, reality environment. This system was developed in Cognitive Informatics Group of MTA SZTAKI research institute, more information about it can be found under [6]. In such case when the industrial robot working in dangerous environment the user personally presentation is expensive and unnecessary risk. In this case the controlling the device with a method, which is close to real manner is a good solution too. So the VIRCA connects research groups and distant laboratories (devices) over the Internet using standard protocols. The main idea of VIRCA is placing the physical device in a generated virtual space , as you can see in Figure 3.

All places connect to a name server to register their device list. When the components are up and registered on the network, the operators can choose their input and output devices (cognitive informatics equipment and robots) and use them to solve the given task.

The main goal is to develop a system which is possible to visualize physical devices, connect to them and control them, make a connection and establish interaction among real and virtual objects. Secondary goal is enable more virtual space to connect with each other.

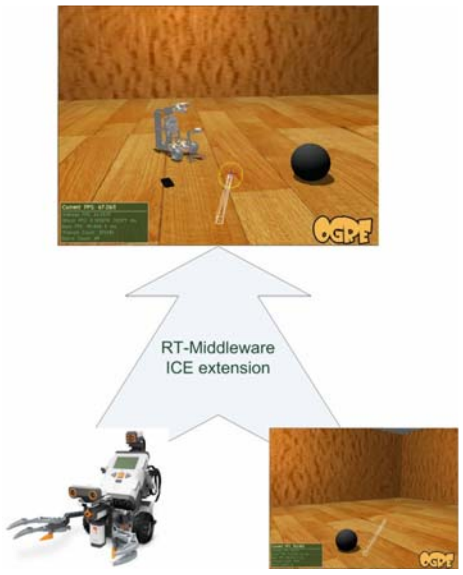

Figure 3
The concept of VIRCA system.

In this system the following mandatory components are needed:

- 3D Space: One 3D visualisation component which continuously renders the virtual word. The devices can register into that virtual space and then it the mesh (model of physical device) is appearing the appropriate position. The virtual object "feel" the outer object presentation (e.g. virtual ball acts in harmony with the physical one). The user can send a command to a physical device via graphical (3D) interface. The 3D Space component has two interfaces:
    - Register interface (provider): The physical (cyber) devices can register/unregister themselves into the virtual world, or send their new positions.
    - Commander interface (consumer): Sends commands to the robot: e.g. move to another position or take something or do a job, etc.
- Cyber devices: Several different physical robot devices can exist at the same time in different places. Each type of robot has its own component which can exist several instances in the same time. In the first step we support two devices: Lego NXT robot and KUKA industrial robot. The cyber device component has two interface:

  - Commander interface (provider): Gets commands from 3D components or any other component, which want to control them.
  - Register interface (consumer): registers into / unregisters from virtual space.

The optional components can be the followings:

- Camera component: provides pictures or movie about a physical space.
- Observer component: gets the picture, which is provided by camera component, identifies the objects and sends its position to 3D engine
- Controller components: variety of form can be existing. The user sends command to the cyber device via this component by his voice or arm or gestures. The input of these components can be provided by camera or microphone or any other cognitive source.

If the system uses more cyber devices then the Commander interface of 3D system consumes more Commander interface of more cyber devices, so the imporved system needed.

Conclusions

The OpenRTM system with the ICE extension is an efficient solution for building distributed robot systems containing more parts running on heterogenic platforms. The suggested improvement provides a fast negotiation among components and supports the connection among more providers and one cunsumer in a transparent way. The first step of building a system is design of the components and their interfaces. If the parts of the new robot system communicates via well defined interfaces, it can be extended further in a rather easy way.

Acknowledgement

The research was supported by NAP project NKTH-KCKHA005 (OMFB-01137/2008).

References

[1] http://www.omg.org/robotics/

[2] A New Approach to Object-Oriented Middleware, IEEE Computer Society 2004

[3] http://www.openrtm.org/

[4] Gabor Sziebig: Sziebig Gábor, Gaudia Andor, Korondi Péter, Ando Noriaki, Solvang Bjørn, Robot Vision for RT-Middleware Framework, In Proc. IEEE Instrumentation and Measurement Technology Conference (IMTC'07), pp. 1-6, 2007

[5] Gábor Sziebig: Sziebig Gábor, Gaudia Andor, Korondi Péter, Ando Noriaki, Video image processing system for RT-middleware, In Proc. 7th International Symposium of Hungarian Researchers on Computational Intelligence (HUCI'06), pp. 461-472, 2006

[6] http://www.sztaki.hu/department/Cogvis/