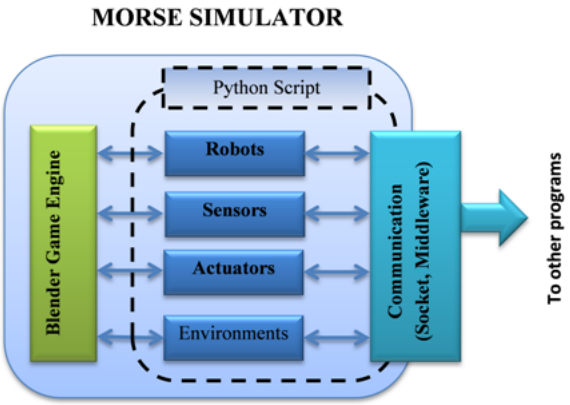


Figure 1: MORSE Architecture

runs based on the specification script in Python written by the user. The user needs to specify robots in the script, sensors and actuators attached to each robot, communication or middleware type for communicating with other programs, and the environment to be used.

The MORSE calls the model of each object and appends them to the Blender Game Engine, then reads or writes the data of objects according to their behavior inside Blender. Lastly, the MORSE will pass the data to middlewares or communication section to be used by external programs.

3. Integrating RT-Middleware and MORSE

RT-Middleware is a CORBA-based infrastructure software implemented using a number of specifications at the distributed middleware interface level, authorized by Object Management Group (OMG) [3]. The implementation of this middleware we use is called OpenRTM-aist. This middleware aims to build a modular structure of robots and their parts (such as sensors and actuators) at the software level and to ease the process of building robots by simply combining modules. It allows system designers to effectively build customized robots for many applications. The component modules used to construct robotic systems are called RT-Components.

3.1. RT-Middleware Support for MORSE

The reason of integrating RT-Middleware and MORSE is to extend RT-Middleware support for many external libraries or applications. Another reason is that there are many RT-Components (RTC) which perform various algorithms for robotics, such as SLAM, motion planning, mapping, people tracking, etc. It is a good idea to test those algorithms in a simulator before applying them to the real robot. A realistic simulator like MORSE can help us to make a simulation as close as the real situation of the robot. It encourages us to to make RT-Components that will be the bridge of existing algorithms which have been applied as RTCs, and the MORSE Simulator (see Fig. 2).

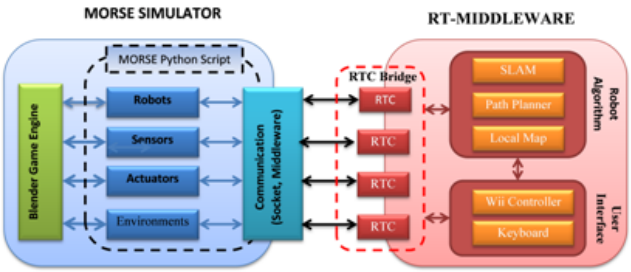


Figure 2: Integration of MORSE Simulator and RT-Middleware

3.2. RT-Component for MORSE

Basically, we create RT-Components as bridges between MORSE simulator and other RTCs in the RT-Middleware environment. Our RTCs exploit the MORSE to realistically provide data of the robot and its sensors. Since RTCs and MORSE have a great sense of modularity, we can easily exchange them with the real robot or sensors module for the real experiments, without changing other components. As the interface to the existing RTCs, we adopt the one defined by the mobile robot Sub-WG of NEDO Intelligent RT Software project. We implement RTCs for MORSE simulator in two methods:

3.2.1. User-defined RTC

This method allows us to define components and its data organization. For example, we can separate simulated laser range finder as one module to imitate Top-URG module (a component for retrieving Hokuyo laser data), while we can merge simulated pose sensor, velocity sensor, and motion controller as one component to imitate the real robot controller component (see Fig. 3a).

This method utilizes socket for communication between MORSE and RTCs. We have to retrieve port address of each sensor and actuator assigned by the MORSE and configure port address of RTCs manually. The workflow is as follows:

1. Create MORSE script defining robots, sensors, actuators, environment, and communication types;
2. Run the MORSE script;
3. Run RTCs in accordance with objects we use in MORSE script;
4. Configure the port communication of each RTC;
5. Connect with external algorithms (e.g. SLAM, path planner, etc.).

Currently, we have made several components for bridging robot controller, camera, stereo camera, laser range finder, pan tilt unit, and artificial human component, using C++ version of OpenRTM (see Table 1). We also have made 3D model for PeopleBot and human posture.

3.2.2. Automated RTC Generation

This method creates RTCs directly from the MORSE Python script. We add OpenRTM definition inside

Table 1: Development Environment for Integration of MORSE and RT-Middleware

Software	User-defined RTC	Automated RTC
OS	Ubuntu 10.04	Ubuntu 10.04
MORSE	0.5.2 Stable	0.5.2 Stable
Blender	2.59	2.59
Python	2.6 (OpenRTM) 3.2.2 (MORSE)	2.6 (OpenRTM) 3.2.2 (MORSE)
RTM	OpenRTM-aist (C++) 1.0.0	OpenRTM-aist (Python) 1.0.0
Additional	-	PyRTSeam

MORSE core (we call it OpenRTM extension script), and modify the middleware part of MORSE to allow us generate the RTC for each sensor and actuator defined in the MORSE script (i.e., one RTC for one sensor). All of component's naming and connection are taken care automatically through Python script (see Fig. 3b).

We create RTC templates for each sensor and actuator using PyRTSeam [6]. When we define the robot and sensors we use in the MORSE Python script, the OpenRTM extension will read the name and connection of each robot and sensor directly inside the MORSE environment, then passes it as arguments to RTC templates. As we run the MORSE script, RTCs are automatically created.

We have made RTC templates for pose sensor, velocity sensor, laser range finder, motion controller, and pan tilt unit, for using this method.

4. Implementation

We have implemented integration of RT-Middleware and MORSE for several applications:

- **Path Planning Test**
  We use MORSE simulator and RT-Components for testing our path planning algorithm [2]. This simulation utilizes Local Map RTC, Waypoint Sender RTC, Image Viewer RTC, and Path Planner RTC (see Fig. 4b for path planner usage in a complex system). The simulated robot follows waypoints given by Waypoint Sender RTC, while avoiding obstacles.

- **Indoor Exploration**
  We demonstrate the exploration algorithm using MORSE and RT-Middleware integration. This simulation gives an example how the MORSE-RTM integration works on a complex system, involving SLAM, localization, path planner, and exploration components. The simulated robot explores the simulated environment representing our ICT building of Toyohashi University of Technology (Fig. 5). The environment is made for imitating the real building, as well as its furnitures (see Fig. 4a).

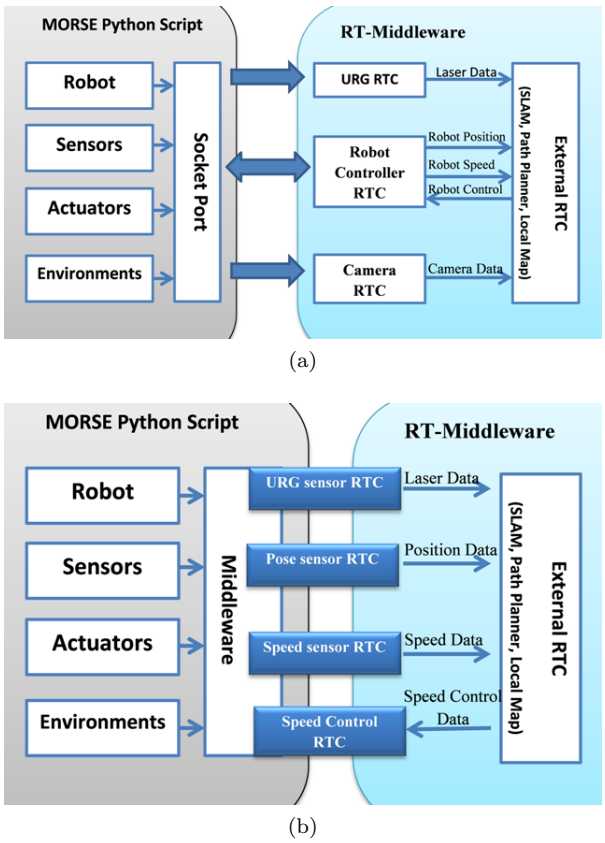


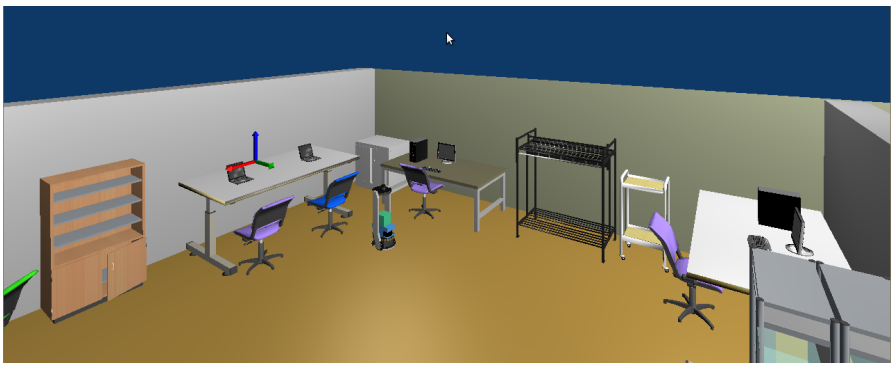
Figure 3: RTC implementation methods for MORSE: (a)User-defined RTC method, (b) Automated RTC generation method.

- **Viewpoint Planning for Attendant Robot**
  It is another complex system example for using MORSE-RTM integration. This system is our current research, where the robot always watches a specific person from selected viewpoints. This system involves simulated camera performing object tracking task inside the MORSE simulator (see Fig. 4c). It also uses several RTCs such as viewpoint planner, path planner, localization, and local map components.

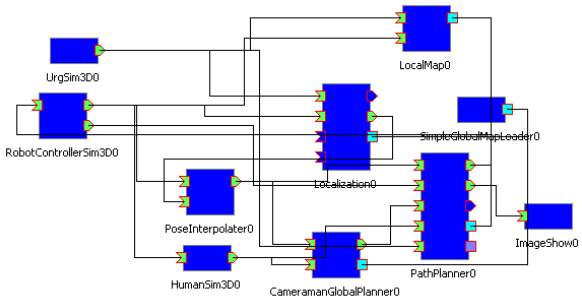
- **Remote Control for Real and Simulated Robot**
  We show how an external controller for the real robot can also be used for simulation in this MORSE-RTM integration (see Fig. 4d). We have already used keyboard controller RTC and Wii remote controller RTC for controlling both real and simulated robot. The user can directly use the controller RTC to both real and simulated robot RTC by simply exchanging the connection. Controller RTC will send the velocity data to the MORSE simulator based on the user's command.

5. Conclusion

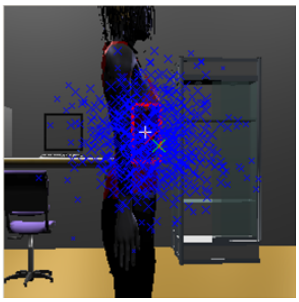
We have presented integration of RT-Middleware and MORSE Simulator using our RT-Components. Realistic simulation can be done in MORSE simulator using the robotic algorithm from RT-Components. Several imple-



(a)



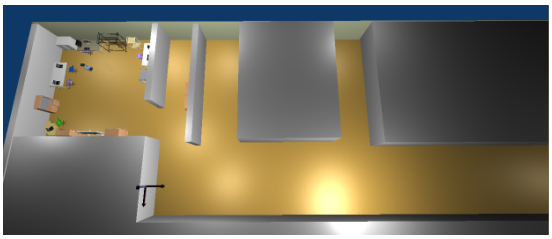
(b)



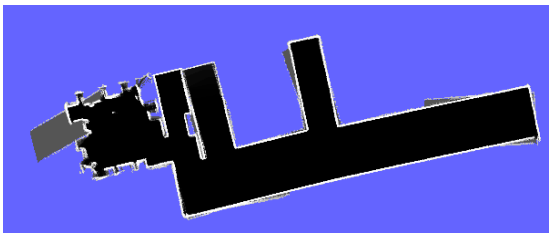
(c)

(d)

Figure 4: Implementation of RT-Middleware and MORSE integration: (a) Simulated environment screenshot created by MORSE, (b) An example of combination of RTC bridge for the MORSE and other RTCs in a complex system, (c) Person tracking task in MORSE simulator, (d) Robot control example using Wii remote controller.



(a)



(b)

Figure 5: Exploration algorithm using RT-Middleware and MORSE integration: (a) Simulated Map of ICT building (see the text), (b) Exploration map result.

mentations have been provided to proof the benefit of RT-Middleware support for MORSE simulator.

In the future, we plan to add list of RT-Components for supporting sensors and actuators in the MORSE. We also want to implement other scenarios for different robots and environments.

References

[1] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan. "Modular Open Robots Simulation Engine: MORSE". In Proc. of IEEE Int. Conf. of Robotics and Automation (ICRA), pp. 46-51, 2011.

[2] I. Ardiyanto, J. Miura. "Heuristically arrival time field-biased (HeAT) random tree: An online path planning algorithm for mobile robot considering kinodynamic constraints". In Proc. of IEEE Int. Conf. on Robotics and Biomimetics (ROBIO), pp.360-365, 2011.

[3] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, W.K. Yoon. "RT-middleware: Distributed component middleware for RT (robot technology)". In Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3555-3560, 2005.

[4] http://www.openrobots.org/wiki/morse/

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

[6] http://www.sec.co.jp/robot/pyrtseam/index.html