A Simulation Environment for OpenRTM-aist

Ian Chen, Bruce MacDonald, Burkhard Wünsche
University of Auckland
New Zealand
{i.chen, b.macdonald}@auckland.ac.nz
burkhard@cs.auckland.ac.nz

Geoffrey Biggs, Tetsuo Kotoku
Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology
Tsukuba, Japan
{geoffrey.biggs, t.kotoku}@aist.go.jp

Abstract — Unified testing of multiple heterogeneous robotic software components is a challenging problem and many robotic systems rely on vendor-specific tools for testing and evaluation of individual subsystems. The consequence is often the unexpected interactions between components that arise during system integration.

OpenRTM-aist is a distributed software framework that standardises the development of robotic systems while encouraging software reuse and improving the efficiency of the system integration process. The problem is the lack of a well-integrated simulation tool that provides a safe, virtual test environment for evaluating OpenRTM-aist components.

This paper presents a simulation environment for OpenRTM-aist. As opposed to creating a built-in simulation tool tied to the OpenRTM-aist architecture, we use an existing general purpose robot simulator, namely Gazebo, because of its modular design and framework independent architecture. We show that by creating an interface layer to Gazebo, robotic systems developed using OpenRTM-aist can be tested in Gazebo simulation without modifications to the underlying software code. In addition, we demonstrate the interoperability between OpenRTM-aist component-based robot systems and Player client programs in achieving a global robot task in the same simulation context.

Keywords: *OpenRTM-aist, Robot Simulation*

I. Introduction

As technology advances, robotic systems are being built from an increasing number of hardware devices, equipped with a diverse range of sensors, and controlled by complex software algorithms. The problem of integrating multiple robot components is one that is common to many robot developers across different application domains. As a result, guidelines and standards have been devised in the robot community to promote reusability and interoperability.

The Object Management Group [1], a large consortium of companies and research institutions, develops open standards for a range of technologies. In recent years, it has begun developing standards for robotics. The first standard produced is the Robotic Technology Component standard [2], which specifies the requirements for component-based robot software, including interfaces and execution semantics. The standard is itself an extension of the Super Distributed Objects standard.

OpenRTM [3] is a distributed software framework based upon this standard. It complies with the standard, as well as providing additional features, such as well-defined data interchange semantics and component callbacks for asynchronous execution. OpenRTM-aist [3] is the open source implementation distributed by the National Institute of Advanced Industrial Science and Technology (AIST).

As the OpenRTM community continues to grow, it is in need of a simulation environment for safe and cost-effective testing of OpenRTM based mobile robotic systems that is simple to use directly from OpenRTM. Robot simulation is an important part of the robot development cycle for creating robust, high quality robotic software, and can provide useful insights to potential real world problems. There exist many simulation tools specific to robot vendors that produce accurate simulation results for particular robot components. However, robot system integration requires a simulation environment capable of performing unified testing of multiple robot components integrated in one application.

Where a diverse range of robotic components and systems is controlled by the same framework, such as the case with OpenRTM, there is an additional advantage to an integrated simulation tool, which can provide additional cost savings by close integration with the control architecture as well as a variety of components and vendors. Ideally the simulation environment should provide the same software interface as the real world devices, so that OpenRTM-aist applications may be conveniently tested with both simulated and real scenarios.

This paper presents a general robot simulation environment for OpenRTM-aist. A set of interface components is created that enable OpenRTM-aist robot systems to exchange data with an existing 3D robot simulator in the same way as it would with real robot devices.

Section II describes related work. Section III describes the OpenRTM-aist framework. Section IV describes the selected robot simulation tool. Section V details the implementation of the simulation interface layer that links OpenRTM-aist to the robot simulation tool. Section VI presents results from simulation of various robot tasks, and Section VII concludes the paper.

II. Related Work

There are a number of publicly available robot simulators with large community support. These simulators are designed to be generic and extensible to meet the needs of the wide range of users.

The Microsoft Robotics Studio simulator [4] is a 3D robot simulator that offers high fidelity simulation of general robot tasks. The simulator is built on a service-oriented architecture, and each entity in simulation provides a number of operations in the form of a service for exchanging messages with other components. The main limitation is the closed source nature of both the simulator and the underlying Agei PhysX [5] physics engine.

USARSim [6] is a 3D mobile robot simulator, initially designed for simulation of urban search and rescue operations, and is commonly used as the research simulation platform in the RoboCup [7] community. The simulator is built on the commercial game engine, Unreal Engine 2 [8], and controls to simulation entities in the engine is done through the provided Gamebot interface. While the simulator itself is open source, the game engine is proprietary.

OpenRAVE [9] is an open source planning and simulation tool. The plug-in driven architecture supports the integration of custom functionalities, such as planning, control, or sensing modules, that are loaded at run time. Interaction with the simulation uses high level scripts in scripting environments such as Octave or Matlab. The simulator is designed to be cross-platform, and many of the components are reusable. However, the focus of the simulator has been placed on robot manipulation and humanoid robot tasks.

There is a robot simulation tool in the OpenRTM community, available from the OpenHRP project [10]. The OpenHRP dynamics simulator simulates humanoid robotics and is used for development of humanoid software applications. The simulator uses Virtual Reality Modeling Language (VRML) for modelling and features sophisticated algorithms for dynamics simulation, contact force calculation, collision detection, and walking pattern generation. OpenRTM is already able to interact with the OpenHRP simulator. However, this simulator is focused in particular on humanoid robots. The design of the simulator is also intertwined with OpenHRP's architecture and separation of the simulator from OpenHRP to work with existing RT Components from other projects is difficult. What OpenRTM needs is a simulator that handles the suitable mobile robot simulation problem well, and is simple to use directly from OpenRTM. Without one, OpenRTM will be limited in the range of robotic systems it can accurately and completely simulate. This paper presents an alternative robot simulator for OpenRTM-aist that is simple and light-weight. We build on an existing robot simulator designed for simulation of general robot tasks.

III. OpenRTM-aist

OpenRTM-aist is based on the CORBA (Common Object Request Broker Architecture) distributed object middleware and implements the RT Component standard. RT Components form the basis of an OpenRTM-aist system.

Components communicate via two types of ports. Data ports provide one-way, asynchronous communication. An *InPort* subscribes to an *OutPort*, which publishes data. Service ports are used for request-response communication. Component introspection is supported, in keeping with the standard, which allows a range of component metadata to be discovered at run time, such as component capabilities and available ports. Internally, components execute a state machine. States include *activated*, *deactivated*, *execute*, and *error*.

OpenRTM-aist provides two graphical tools for developing robot systems. RTCBuilder is an Eclipse plug-in for generating the skeleton code of RT Components based on user-provided specifications. RTSystemEditor is another Eclipse plug-in used for connecting multiple RT Components to create a complete robot system, by linking data and service ports between components.

IV. Robot Simulation Technology

Redesigning the OpenHRP architecture for interfacing directly with other RT Components is one option for creating a general purpose OpenRTM simulator. However, substantial effort and time would be necessary.

There are a number of general purpose robot simulators with large community support as reviewed in Section II. Instead of taking the time-consuming process of building a robot simulator from the ground up, we take advantage of existing technology and choose from the range of readily available robot simulator as the base simulation platform for OpenRTM-aist.

The Gazebo robot simulator has been chosen based on its functional features, simulation fidelity, flexibility and ease of integration, extensibility, and documentation and support. An open source project is preferred as it conforms to OpenRTM-aist's goal for an open architecture.

A. Gazebo

Gazebo is an open source 3D robot simulation tool widely supported and used by many research organisations. Gazebo is part of the Player Project [11] for development of robotic and sensor applications and adopts a similar architecture design as Player. It features a publisher/subscriber, a form of client/server, model of communication. The design of Gazebo enables the simulation platform to work seamlessly with Player robot systems, and in addition, the operation of Gazebo is independent of Player's framework, which greatly increases the reusability of the simulation tool.

Gazebo uses open source rendering and physics libraries that also have wide community support. The Open Dynamics Engine (ODE) [12] is used to provide rigid body physics simulation and OGRE3D [13] provides high quality 3D graphics rendering. Each simulation entity in Gazebo can be associated with zero or more `controllers` that handle commands for controlling the associated simulation entity, as well as generating data describing the states of the entity.

The core of Gazebo's simulation platform is not designed to provide network communications, as this should be the job of robot middlewares such as Player. Gazebo enables clients to access its data using shared memory (SHM). Data generated by `controllers` are published using Gazebo interfaces, known as `Ifaces`. The data is written to a shared memory space where it can be read, also using `Ifaces`, by other processes. This enables interprocess communication between the client robotic software and Gazebo, independently

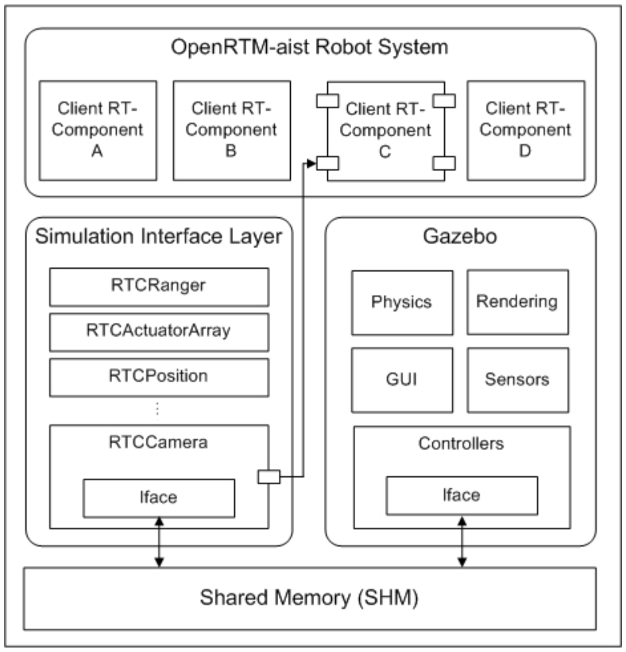


Fig. 1. The data flow between client RT Components, interface RT Components and Gazebo. As an example, data is shown being sent from the InPort of a camera interface RT Component to the OutPort of a client RT Component. The InPorts and OutPorts of other RT Components are not shown for clarity.

of the platforms, frameworks, or programming languages. The C++ library `libgazebo` is provided by Gazebo for communication with the simulation data stored in SHM.

V. Simulation Interface Components

To link OpenRTM-aist with Gazebo, a set of interface components has been developed. The components encapsulate the underlying simulation platform and enable client RT components to exchange data with Gazebo without modifications to the client's software code.

For each type of simulation device in Gazebo, such as a sensor device (a laser range finder or a camera, for example), a corresponding RT Component is created; we will refer to it as an interface RT Component. The interface RT Components are responsible for exchanging data between `controllers` in Gazebo and the client RT Components. An interface RT Component can have zero or more InPorts that accept commands sent from client RT Components. There can be one or more OutPorts depending on the types of data each simulation device is capable of generating. The simulation framework is shown in Fig. 1.

The life cycle of an interface RT Component is as follows:

- Initialisation – The interface RT Component is initiated with a default Gazebo server ID, and simulation device ID that it will subscribe to. These values are can be configured using RTSystemEditor.
- Activation – The interface RT Component subscribes to one simulation device using `libgazebo`.
- Execution – The interface RT Component continuously monitors for new data published by the subscribed simulation device. Incoming data are processed into robot data formats consistent with the RT Component's interface guidelines [3]. The resulting data are then forwarded to the client RT Components through its OutPorts. Service requests from client RT Components are handled in a similar manner.
- Deactivation – The interface RT Component unsubscribes to the simulation device.

RTSystemEditor is used to create connections between the interface RT Components and the client RT Components. In cases where there are multiple simulation devices of the same type in Gazebo, such as two laser sensors, instances of the an interface RT Component can be created and configured in RTSystemEditor to map to the corresponding device based on its unique ID. The configuration of RT Components can be saved as XML files which can later be imported back to RTSystemEditor to restart the simulation.

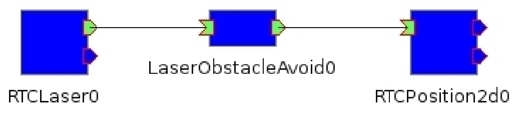
VI. Results

A number of interface RT Components have been implemented in C++ for OpenRTM-aist-1.0-RC1. We show results of three simulations of OpenRTM-aist robot systems in Gazebo.

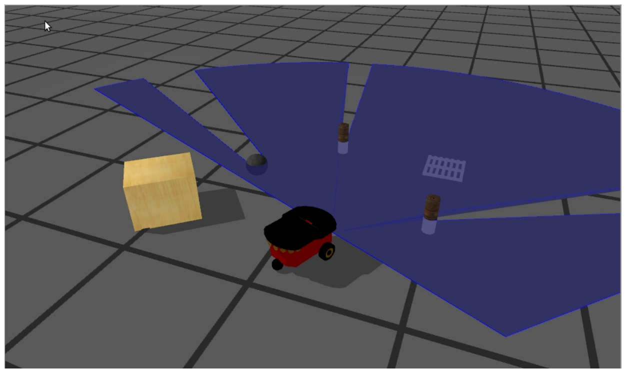
An obstacle avoidance RT Component has been created and simulated in Gazebo.[1] The algorithm controls a robot to navigate towards the open space while avoiding obstacles detected by the laser range finder. Two interface RT Components are used. A laser interface RT Component reads range data from SHM then forwards the data through its OutPort to the InPort of the obstacle avoidance RT Component. The obstacle avoidance RT Component uses the range data to determine the appropriate velocity and turn rate that the robot should move at and sends the commands to the position2d interface RT Component. The commands are then communicated to the Gazebo simulation device controller over SHM. In simulation, we use the available Pioneer2DX model provided by Gazebo as the primary robot platform and equip it with a simulated Hokuyo URG laser range finder. Fig. 2(a) shows the RT Components used and the layout in RTSystemEditor. Fig. 2(b) shows the Gazebo simulation.

The second example shows a video player RT Component simulated in Gazebo. The video player displays image frames captured by a camera device. A camera interface RT Component reads image data from SHM then sends the data through its OutPort to the InPort of the video player RT Component. In simulation, we mount a camera sensor on an unmanned helicopter model that overlooks an agricultural environment consisting of a moving cow; this simulates an animal monitoring application in agriculture. Fig. 3(a) shows the components in RTSystemEditor, and Fig. 3(b) shows the Gazebo simulation and the video player RT Component displaying the view captured by the onboard camera.

[1]The obstacle avoidance algorithm has been ported from the Player Project.

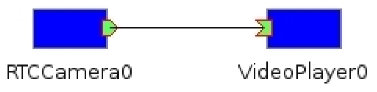


(a)

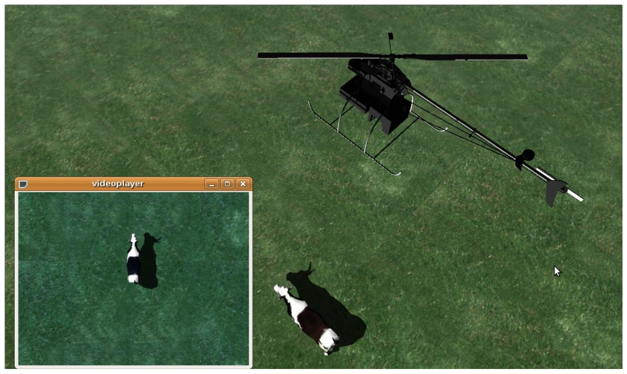


(b)

Fig. 2. (a) The RTSystemEditor showing the connections between the laser interface RT Component (RTCLaser), the obstacle avoidance RT Component (LaserObstacleAvoid), and the position2d interface RT Component (RTCPosition2d). (b) A screenshot showing the Pioneer2DX robot avoiding various obstacles in Gazebo



(a)



(b)

Fig. 3. (a) The RTSystemEditor showing the connection between the camera interface RT Component (RTCCamera) and the video player RT Component (VideoPlayer). (b) A screenshot showing the agricultural simulation environment consisting of a helicopter robot and a cow. The robot is equipped with a downward looking camera that captures images displayed in the window shown on the left.

The last example demonstrates the interoperation of an OpenRTM-aist robot system and a Player client robot system in the same Gazebo simulation instance. The example shows two Pioneer robots in a maze-like environment navigating in a leader-follower formation. The obstacle avoidance RT Component is used to control the lead robot to explore the environment while the Player client program controls the second robot to follow closely behind. Fig. 4 shows a screenshot of the Gazebo simulation. The result shows the flexibility of Gazebo to act as a single, shared simulation environment for testing robot systems created using different robotic software frameworks.

The ease with which OpenRTM-aist was linked to Gazebo shows the usefulness of an approach to robot software development that is modular with defined interfaces. Gazebo is part of the Player Project and was designed to work well with Player. Had Gazebo been integrated in the Player Architecture, linking OpenRTM-aist would have been more difficult. It would also have introduced inefficiency by requiring communication through a separate framework in between OpenRTM-aist and Gazebo. The separation of functional (Gazebo) from integration (Player) software has, in this case, greatly increased the reusability of the functional software.

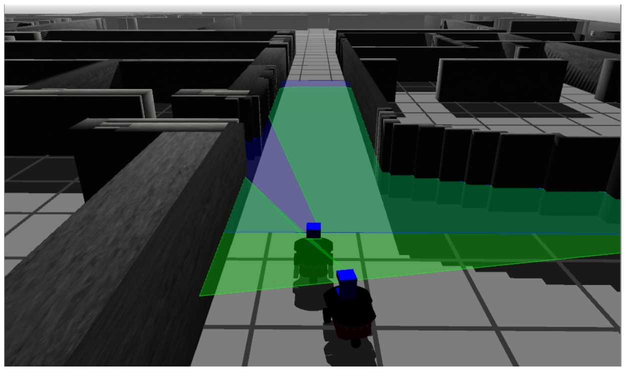


Fig. 4. A screenshot showing an OpenRTM-aist controlled robot (leading robot with blue laser scan) working with a Player controlled robot (following robot with green laser scan) in Gazebo. Both Pioneer2DX robots are equipped with a simulated SICK LMS 200 laser range finder.

VII. Conclusions

We have presented the design and initial implementation of a 3D virtual robot simulation environment for robotic systems that use OpenRTM-aist. A set of interface components has been implemented that enable communication between RT Components and the Gazebo simulation platform. We have also demonstrated the usefulness of the simulation tool on three simple tasks.

Future work includes improving the simulation interface components for a more complete support of the various types of robot devices based on the RT Component interface guidelines. The simulation interface components will also need to be tested with larger OpenRTM-aist robot systems.

Acknowledgment

This project is financially supported by the National Institute of Advanced Industrial Science and Technology, and the University of Auckland Doctoral Scholarship.

References

[1] "The Object Management Group (OMG)," October 2009, http://www.omg.org/.

[2] "Robotics domain task force," http://robotics.omg.org/.

[3] National Institute of Advanced Industrial Science and Technology (AIST), "RT-Middleware: OpenRTM-aist," 2009, http://www.openrtm.org/.

[4] Microsoft, "Microsoft robotics studio," 2009, http://msdn2.microsoft.com/en-us/robotics/default.aspx.

[5] Ageia, "Ageia PhysX," 2009, http://www.ageia.com/.

[6] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference on Robotics and Automation*, Roma, April 10-14 2007, pp. 1400–1405.

[7] The RoboCup Federation, "Robocup," February 2008, http://www.robocup.org/.

[8] Epic Games, "Unreal Technology," 2009, http://www.unrealtechnology.com/.

[9] R. Diankov and J. Kuffner, "OpenRAVE: A planning architecture for autonomous robotics," Robotics Institute, Carnegie Mellon University, Pittsburgh, Tech. Rep., 2008.

[10] "OpenHRP - Open Architecture Humanoid Robotics Platform," http://www.openrtp.jp/openhrp3/en/index.html.

[11] Player/Stage, "The player/stage project," 2009, http://playerstage.sf.net/.

[12] R. Smith, "Open dynamics engine," http://www.ode.org/.

[13] OGRE, "OGRE 3D : Object-oriented graphics rendering engine." http://www.ogre3d.org.