Proceedings of IC-NIDC2009

DEVELOPMENT OF A ROBOID COMPONENT FOR PLAYER/STAGE ROBOT SIMULATOR

Jun Won Lim[1], Sanghoon Lee[2], Il Hong Suh[1], and Kyung Jin Kim[3]

[1]Dept. Of Electronics and Computer Engineering, Hanyang Univ., Seoul, Korea
[2]BK21 Advanced IT Education Program on Industrial Demand, Hanyang Univ., Seoul, Korea
[3]Robomaion, Co., Ltd. Seoul, Korea

jwlim@incorl.hanyang.ac.kr, shlee@incorl.hanyang.ac.kr, ihsuh@hanyang.ac.kr, jin_khim@hanmail.net

Abstract

Simulation environment is very useful as it provides ground to virtually embody robots in the designing stage. There will be applied Player/Stage, one of the robot simulation and control environments, to Roboid Studio which is a component-based framework. Here, robots are controlled based on simulacrum. Several issues will be discussed to get Player/Stage to work as a component for Roboid Studio.

Keywords: Robot Simulator; Robot S/W Platform; Robot S/W Component; Roboid; Simulator

1. Introduction

Robot-applied education is broadly active from kindergarten kids to adults. Korean domestic robot market is being made according to this bigger stream. Accordingly, demands for education robots are increasing, and various robots especially such as Intelligent Robot Practice Kit, programmable robot Mindstorm NXT by Lego, Bioloid of Robotis are widely adopted. Korean ministry of Knowledge and Economy has been active in developing intelligent robots, and recently announced its goals of reaching 4 trillion won's robot industry market capitalization and increasing global market share to 13.3% both by 2013. Especially education robot sector expects 7.7 millions of clients solely in Korean domestic elementary, middle and high schools.

In Robot-applied education, the realization of algorithms meaning the actual design of robot software has further educational meaning, than the simple assembly of hardware. For this algorithm design, we need a robot software platform that allows robot software production to non-professionals, as well as to developers. We utilized Roboid Studio [4] which supports various contents production tools.

Roboid Studios is an Eclipse-based framework to develop applications for Thin-Client Network Robots. This framework assists programs and has visual interface, code library, modeling language, script language and communication protocol. Roboid Studio supports various contents production tools thus allows ordinary users to produce and design robot contents and software modules. Various robot control and service, such as current Lego NXT, Pelicanoid, camera and mouse have been produced with Roboid Components, with its component-based platform.

However, although the tool has allowed easier robot software programming, it is not capable enough to satisfy the needs of users to control more various types of robots. To actually develop robots, a lot of efforts as well as time are consumed, and the risk of failure is also high. To relieve this, simulation environment is widely adopted, and we adopted Player/Stage [6] in this thesis.

Player/Stage is two individual projects. Player project is a software project developed to control various robots based on TCP/IP. Stage is a simulator to create robot, sensor and circumstances and simulate through the environment through Player.

We apply Player/Stage as a component of Roboid Studio to utilize benefits of components and various supportive functions of Roboid Studio to provide opportunities of controlling several types of robots, as well as easy development of robot software. Section 2 explains the modeling method of Roboid component. Section 3 shows the benefits and utility of Player/Stage component through experiments.

2. Roboid component design of Robot simulator

2.1 Component of Roboid Studio

2.11 Software component

Software component is a group of software that each can be independently executable, and each equipped with standard interface, substitutability, reusability and functional independence. As well, this also means a part of software[3] which is

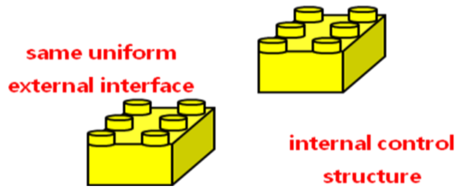


Figure 1. Software component

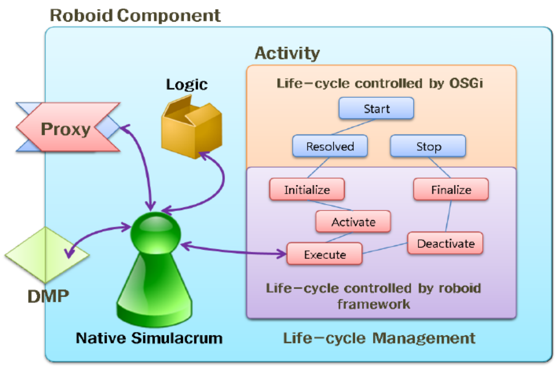


Figure 2. Life-cycle of Roboid component

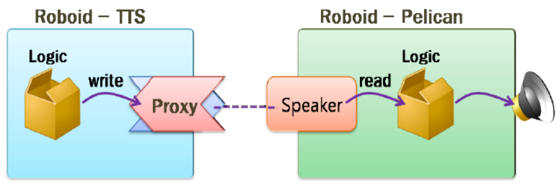


Figure 3. Communication between components

independently developed to implement specific functions, along with well-defined interface to work easily with other components to build a system in accordance with other components[3].
Software component, for the above requirements, provides software infrastructure that is upper-level structured and reusable as well as assembly concept-reusability. Accordingly, they are used in various applications areas. The concept of software component is explained with Figure 1.

2.12 Roboid Component

Roboid component and software component share the same concept. Roboid component, as well, is simple, reusable and functionally independent, and these characteristics provide ease in developing service robots. Each Roboid component is essentially equipped with robot model given by modeling process to make robot component. In robot models, hardware configuration and software device are defined.
Roboid component has 5 life-cycles in total. Initialize when the platform is executed first, Activate for when contents are executed, Execute is executed every 20 ms during contents play, Deactivate to stop the execution, and Finalize to end the platform. These life cycles guarantee compatibility and sync even various components

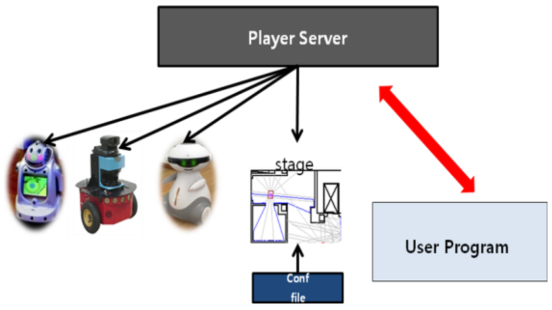


Figure 4. Player/Stage process

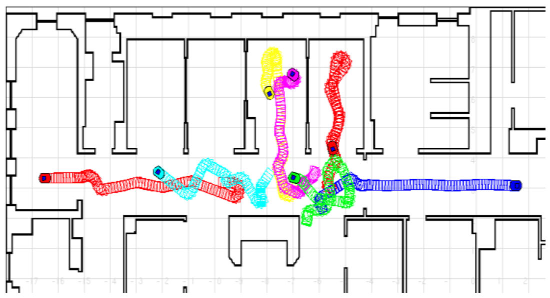


Figure 5. Stage Simulation

compose a single software module. Figure 2 shows life cycles of Roboid Component. When assembling components, each device of robot model file of the component can be connected by proxy type. Figure 3 shows this.

2.2 Player/Stage

Player/Stage[6] is both provided open source and does not require program purchase process. They are software tools for robot and sensor application developed and updated each since 1999 and 1998. Player is a server, connected to the actual robot or to Stage and user application reaches the Player server to control robot and receive sensor data. Figure 4 is regarding this process.
Player is a socket-based device server that provides network interface for various robots, sensor and actuator. Player’s Client/Server model allows the robot control application to be programmed in any programming language, and to make motions in any computer when robot and network are connected. Player also supports several clients to connect the device, and shows a new opportunity of decentralized robot control.
Stage is a simulator which provides simulations of frequent and various robot and sensors based on 2-demensional bitmap environment. It provides various sensor models of Sonar, laser, IR, pan-tilt-zoom camera, blob detection, etc. This has two purposes of first, to provide faster development of controllers in actual robots and second, to provide robot experiments without connecting to actual hardware and environment. Simulation image is shown in Figure 5. As well, this is designed to support multi-agent autonomous system researches. This realizes virtual embodiment of robot, thus

saves the development cost as well as improves the quality of developing robots and applications [5].

3. Meta Component

We are to apply Player/Stage as a component of Roboid Studio. The benefits of the component, such as reusable, easily changeable, easily manageable, quality and maintenance and better productive, and various supports of Roboid Studio are adopted to provide user-friendly simulation.
For this, there are two problems to solve. First, the robot model should be defined through modeling to make Player/Stage as components, and second, API method (vector method) of Player/Stage should be made compatible by being bitmap method [2] of Roboid Studio.
To solve the above 2 problems, there are the following embodiment issues, and we suggest meta component for this.

1) If the user connects Stage to use each sensor and actuator, each device should be created using robot object earned from the first connection. This implies that the robot object creates and manages each device.
2) Meta component has to receive sensor data from Stage. Sensors of the Stage are comparably varied and the interface is standardized, thus the user need to acknowledge that each sensor is connected to what point. Through this, meta component has to activate each sensor and receive the data.
3) In case that each sensor is characterized by the sensor component's name from Initialize, the names are not identifiable if they are created randomly by the user. Naming order needs defined.
4) Sensor can either be faster or lower than the cycle of Roboid Component execution (20ms).

Stage builds the data of environment, robots and sensors to use in the file. Accordingly, it looks as if the base robot manages separable sensors. Player/Stage, as Roboid component, connects and distributes the sensors from the meta component to each sub-component. This implies that meta component is a component of the component, model of the model. Figure 6 shows the meta component.
Meta component, in a big picture, functions in 3 ways. First, it provides unified and flexible interface to each sub-component for easy plug-in. Second, it dynamically distributes and connects sensors on runtime based on the modeling data of each sub-component. Third, it supports assembly to build and control various robots.
Meta component is composed of interface, thesaurus and logic parts. Interface provides dynamic ports to sub-components. Simple drag-and-drop can increase the number of ports, and this enables the communication between meta component and sub-component. Interface part brings modeling data of each sub-component and dynamically creates sensor logic using thesaurus. Thesaurus is a synonyms list that has pre-determined naming order in XML form. In logic part, there are two parts one of which has users' input value-based actuator control function and the other transfers the sensor value dynamically created by the interface and thesaurus. This sensor logic part is composed of threads to manage control periods.
This means that users do not need much as long as they keep the naming order defined in thesaurus, as meta component will control the others.

4. Implementation

4.1 Building simulation environments

4.1.1 Simulation environment

The experiments for the component application proposed in this thesis utilize Windows-based Roboid Studio and VMware-applied Linux / Unix-based Player/Stage. VMware allows the Windows users to use other types of operating systems without actually installing them. For the experiments, VMware 5.5.1 and Ubuntu Linux 8.04 were used to drive Player/Stage.
In Stage, as mentioned before, circumstances can be set up, thus we set the map of our laboratory located in IT Building of HanYang University as bitmap file in the configuration file. This is to reduce the differences from actual robot experiments.

4.1.2 Building a thesaurus

A thesaurus is a database that lists words grouped together according to similarity of meaning. In information technology, specialized thesaurus is designed for information retrieval. They are a type of controlled vocabulary for indexing or tagging purposes [9].
In our system, a role of thesaurus is to provide necessary information to make the Robot sensor be connected with executable sensor component. A thesaurus is implemented by XML. Figure 7 shows a scheme of a thesaurus. Figure 8 provides a process of thesaurus.

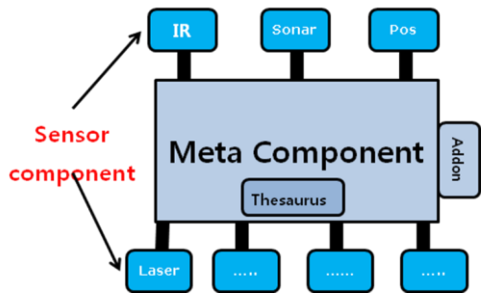
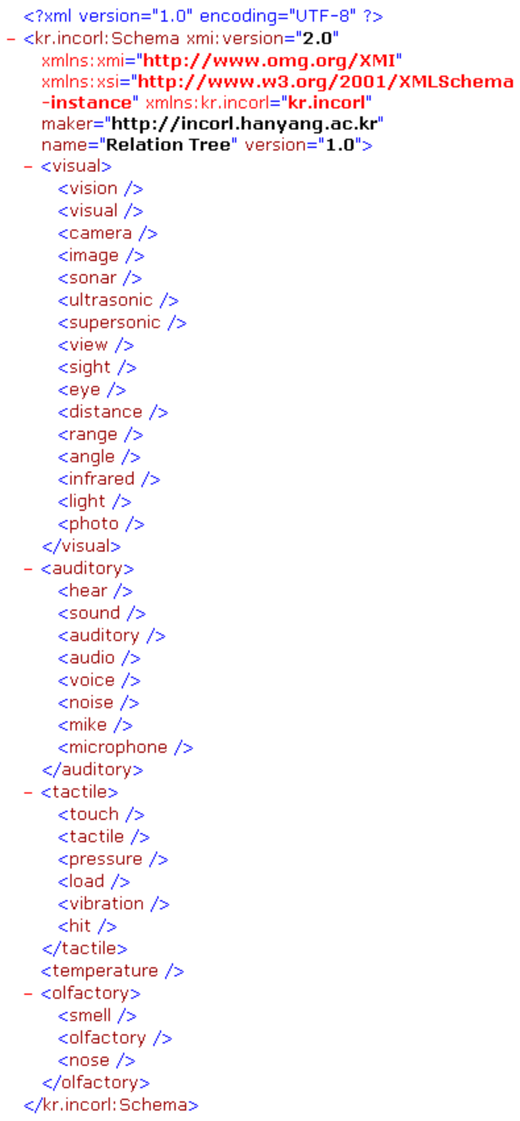


Figure 6. Configuration of Meta component



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <kr.incorl:Schema xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance" xmlns:kr.incorl="kr.incorl"
    maker="http://incorl.hanyang.ac.kr"
    name="Relation Tree" version="1.0">
  - <visual>
      <vision />
      <visual />
      <camera />
      <image />
      <sonar />
      <ultrasonic />
      <supersonic />
      <view />
      <sight />
      <eye />
      <distance />
      <range />
      <angle />
      <infrared />
      <light />
      <photo />
    </visual>
  - <auditory>
      <hear />
      <sound />
      <auditory />
      <audio />
      <voice />
      <noise />
      <mike />
      <microphone />
    </auditory>
  - <tactile>
      <touch />
      <tactile />
      <pressure />
      <load />
      <vibration />
      <hit />
    </tactile>
    <temperature />
  - <olfactory>
      <smell />
      <olfactory />
      <nose />
    </olfactory>
  </kr.incorl:Schema>
```

Figure 7. A scheme of a thesaurus for Robot's sensors

4.1.3 Related library

JavaClient 2.0 [7] was adopted to convert the commands of Player/Stage to bitmap method of Roboid Studio. This enables application development for Player/Stage utilizing the simplicity and function of Java language.

We used JDOM to connect sensor data according to the XML formatted thesaurus of meta component. JDOM is an open source Java-based document object model. It integrates DOM and SAX and supports XPath and XSLT.

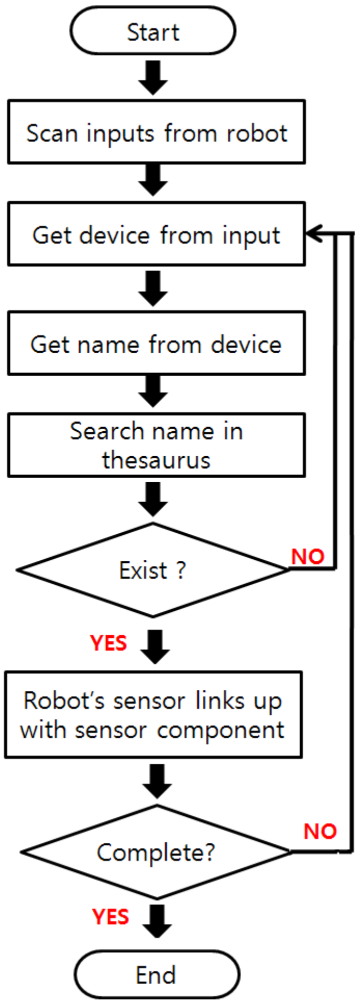
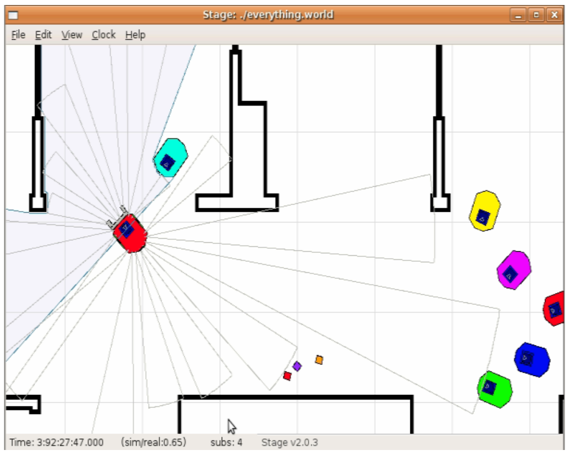


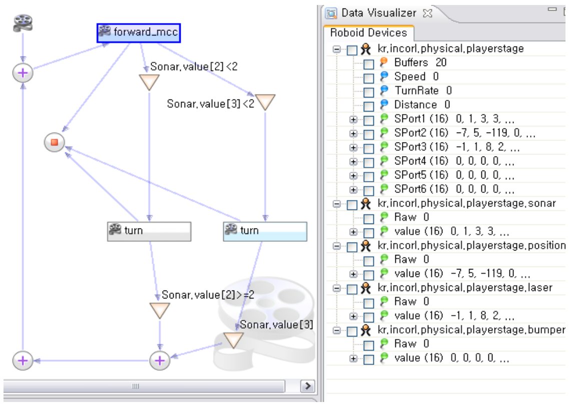
Figure 8. Process of a thesaurus in Player/Stage component

4.2 Simulation

Sensor components to use should be connected to meta component for simulation. It is connected simply by allocating it to the interface part. In this experiment, Sonar, Laser, Bumper and Position sensors were connected to the robot.

There are 2 types of contents development tool based on Roboid Studio; Timeline editor and Motion composer[1]. We test the component we designed to control Player/Stage using the above two. Sonar sensor was used to make obstacle-avoidance autonomous driving program. The result is shown in Figure 9, which went through no problem just like the direct control by programming. Contents were more easily framed compared to the control by Java or C. Figure 10 shows Roboid Studio in simulation. Contents are shown on the left, and simple and intuitive drag-and-drop was adopted.


Stage: ./everything.world
File Edit View Clock Help
Time: 3:92:27:47.000 (sim/real:0.65) subs: 4 Stage v2.0.3
Figure 9. Avoidance autonomous driving program



```
kr.incorl.physical.playerstage
    Buffers  20
    Speed  0
    TurnRate  0
    Distance  0
    SPort1 (16)  0, 1, 3, 3, ...
    SPort2 (16)  -7, 5, -119, 0, ...
    SPort3 (16)  -1, 1, 8, 2, ...
    SPort4 (16)  0, 0, 0, 0, ...
    SPort5 (16)  0, 0, 0, 0, ...
    SPort6 (16)  0, 0, 0, 0, ...
kr.incorl.physical.playerstage.sonar
    Raw  0
    value (16)  0, 1, 3, 3, ...
kr.incorl.physical.playerstage.position
    Raw  0
    value (16)  -7, 5, -119, 0, ...
kr.incorl.physical.playerstage.laser
    Raw  0
    value (16)  -1, 1, 8, 2, ...
kr.incorl.physical.playerstage.bumper
    Raw  0
    value (16)  0, 0, 0, 0, ...
```

Figure 10. Simulation contents execution

Moreover, users can check the sensor data real-time by data visualize shown on the right side.

5. Conclusion

We developed a Player/Stage component as a meta component for Roboid Studio to control simulated Pioneer robot. We asked two first-time Roboid Studio users to develop a system controlling a stimulated Pioneer robot. Averaged hour of successful development was about 3 to 4 hours, where 3 hours were spent to learn how to use Roboid Studio and 1 hour to realize the software logic based on Roboid Studio. This is significantly shorter than the time required for coded simulation control using C++.

The followings are the benefits we could find through componentization of Player/Stage for Roboid Studio.

1) Users can enjoy the benefits of component, such as reusability, easier software modeling by assembling.
2) Development cost is reduced, as both Roboid Studio and Player/Stage are open source projects.
3) Developers can easily make simulation scenario based on various tools to construct contents, and this enables non-professional users to use and control robot simulation.

As a future work, we are planning to support more components usable in Player/Stage and try controlling actual robots (not Stage) in a real environment

Acknowledgement

This work was supported by the Ministry of Knowledge Economy, Technology Innovation Project [2008-F-038-01 Development of Context Adaptive Cognition Technology], and the Second Stage of BK21 Research Division for Advanced IT Education Program on Industrial Demand. All correspondences should be addressed to I. H. Suh.

References

[1] K. J. Kim, I. H. Suh and K.-H. Park, "Roboid Studio: A Design Framework for Thin-Client Network Robots," IEEE International Conference on Advanced Robotics and its Social Impacts, August 23-25, 2008, Taipei, Taiwan.
[2] K. J. Kim, I. H. Suh, S. H. Kim and S. R. Oh, "A novel real-time control architecture for Internet-based Thin-client robot; Simulacrum-based approach," 2008 IEEE International Conference on Robotics and Automation, May 19-23, 2008, Pasadena, USA.
[3] C. Szyperski, Component Software: Beyond Object-Oriented Programming. Reading MA: Addison-Wesley, 1998
[4] www.roboidstudio.org
[5] B. Gerkey, R. T. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” in Proc. 11th Int. Conf. Advanced Robotics, 2003, pp. 317-323.
[6] Player/Stage Project : www.playerstage.org
[7] http://java-player.sourceforge.net/
[8] A. Brooks, T. Kaupp, A. Makarenko, A, Oreback, and S. Williams, “Towards component-based robotics,” in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Aug. 2005, pp. 163-168
[9] http://en.wikipedia.org