Design and Implementation of Tripodal Schematic Control Architecture for Multi-Functional Service Robots

Gunhee Kim, Woojin Chung, Munsang Kim, Chongwon Lee

Advanced Robotics Research Center, Korea Institute of Science and Technology,
39-1 Hawolgok-dong, Sungbuk-ku, Seoul, 136-791, Korea
(Tel : +82-2-958-6743; E-mail: {knir38, wjchung, munsang, cwlee}@kist.re.kr)

Abstract: This paper describes the development of service robotic systems with the Tripodal schematic control architecture. We show practical advantages of the proposed architecture by giving examples of our experience. First, we explain how to add new task using Tripodal architecture approach. The Tripodal architecture provides some crucial organizing principles and core components that are used to build the basis for the system. Thus, the newly developed behaviors, motion algorithm, knowledge, and planning schemes are arranged so as to guarantee the efficiency of the performance of components. Second, we describe the reusability and scaleability of our architecture by introducing the implementation process of the guide robot Jinny. Most of modules developed for former robots like PSR-1 and PSR-2 systems are used directly to the Jinny system without significant modification. Experimental results clearly showed that the developed strategy is useful, even if the hardware configurations as well as software algorithms are more complex and more accumulating.

Keywords: Control architecture, Service robots, Petri nets

1. INTRODUCTION

Three versions of the PSR (Public Service Robot) systems are under development towards indoor public services at the KIST (Korea Institute of Science and Technology). Well-defined control architecture is essential to implement complex robot systems like PSR systems for several reasons. First, control architecture integrates different kinds of hardware and software modules. Second, the architecture plays an important role on maintenance problems such as revision of existing components and addition of new modules. Third, in some cases, functional performance of each module is highly dependent on the architecture. For these reasons, there have been many related research activities so far such as [1][2][3][4][5] [6].

Kim et al. [7] proposed the Tripodal schematic control architecture for the autonomous service robot PSR. The developed control architecture was shown to be a successful framework as verified from the transportation experiments. It provided a good solution to several architectural issues such as information connectivity between a variety of modules, scheduling of information processing, and combination of reactivity and deliberation.

Although transportation tasks are performed successfully with the Tripodal architecture, we realize necessity of extending the capabilities of our systems. Our viewpoint is that the robotic system in daily life should be totally different from the conventional industrial automation problem. The mobile robotic agent should be a multi-functional servant, who can maximally utilize its capability towards various applications. Our major target is to achieve a transportation, a patrol, a guide and a floor cleaning tasks using one robot.

According to the increase of the application domains, the hardware configurations as well as software algorithms are more complex and more accumulating.

Three versions of the PSR have been built, and each of them has its own hardware configuration which has quite different physical properties each other. For example, PSR-1 has a holonomic omni-directional mobile base, a six degree-of-freedom manipulator, a three fingered robot hand, and a reconfigurable trailer system. On the other hand, the Jinny, the newly-built guide robot, has two-wheel differential drive, two arms and 2-DOF neck system for expression of the robot's feelings. The PSR robots also has a variety of sensors, which include encoders, two laser scanners, infra-red and ultrasonic sensors, optical fiber gyros, and force/torque sensors of the hand, potentiometers for measuring trailer orientation, eve-in-hand vision system for object manipulation, trailer-docking vision system, and stereo-vision cameras. We clearly show these diverse hardware modules are incorporated into stable systems with a proposed control architecture approach.

The architecture deals with a wide spectrum of algorithms, which includes from dozens of reactive motion generating behaviors like a compliance controller for opening door and an obstacle-free optimal mobile tracker to several time-consuming algorithms such as localizers, real-time map constructors, and path planners. Moreover, it is required that the architecture should deal with intelligent planning and human robot in order to carry out a guide task.

In this paper, we show practical advantages of the proposed architecture by giving examples of our experience. First, we explain how to add new cleaning task using Tripodal architecture approach step-by-step. The Tripodal architecture provides some crucial organizing principles and core components that are used to build the basis for the system. Thus, the newly developed behaviors, motion algorithm, knowledge, and planning schemes can be systematically integrated without loss of generality. Second, we describe the reusability and scaleability of our architecture. Most of modules developed for PSR-1 and PSR-2 systems are used directly to the Jinny system without significant modification. Only some hardware-related components are changed one-to-one without mal-effect to other modules.

After the brief description of the Tripodal schematic architecture in Chapter 2, the extensibility of the architecture is shown in Chapter 3 through the process of addition of new PSR's functional ability. The reusability of the architecture is proved by porting process from the existing control program to the newly-developed platform in Chapter 4. Chapter 5 illustrates the results of experiments in order to show the efficiency of the Tripodal architecture. Finally some concluding remarks are given in Chapter 6.

2. OVERVIEW OF TRIPODAL SCHEMATIC CONTROL ARCHITECTURE

Tripodal schematic design, as shown in Fig.1, is defined as the architecture of the PSR.

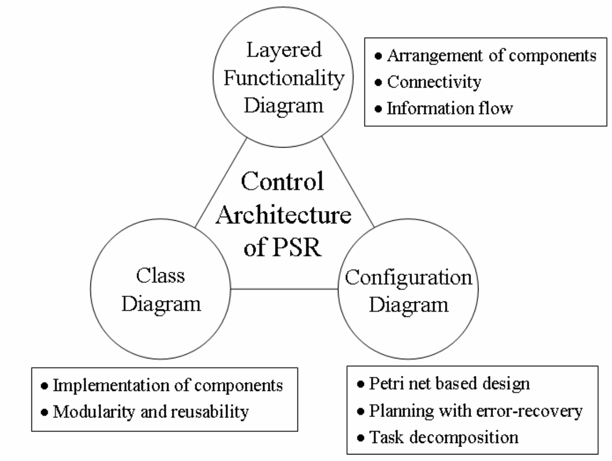
Layered Functionality Diagram
• Arrangement of components
• Connectivity
• Information flow
Control Architecture of PSR
Class Diagram
Configuration Diagram
• Implementation of components
• Modularity and reusability
• Petri net based design
• Planning with error-recovery
• Task decomposition

Fig.1. The overview of Tripodal schematic control architecture

(1) The *layered functionality diagram* is a conceptual diagram for arranging various software modules and functions. It also shows the connectivity and the information flow between components.

(2) The *class diagram* is designed for implementing various types of hardware, software, and functional modules in order to achieve modularity and reusability. It represents instantiation and hierarchy relation between components.

(3) The *configuration diagram* represents Petri nets based configuration design for the planning part of a robot. As described before, the proposed architecture has two types of the configuration, *high-level configuration* and *low-level configuration*, according to the level of layers. .

The detailed description of whole control architecture of PSR is introduced in [7].

3. ADDING A NEW CLEANING TASK

3.1 Task decomposition using *Configuration diagram*

In our architecture, the objective functions of the robot are divided into three levels, a task, a process, and a behavior. The task given by a user is a job which should be autonomously performed by the robot. It includes a transportation, a patrol, a guide and a floor cleaning tasks, in our applications. It is given in form of quite simple command like "Clean the room 3211" or "carry the document box from room 2111 to 2133."

The task is carried out by combination of internal processes generated by the planner. The internal process is defined as a job which is performed by configuring reactive components. The result of the process is definitely divided into success or failure in order to ensure the initial plan is achievable.

Each internal process encapsulates a set of behaviors and related parameters. The behavior is a primitive action that can be executed in very short time.

The planning, the decomposition between the tasks, the processes, and the behaviors, is carried out by referring to configuration. In our architecture, the configuration specifies not only the relation among them but also parameters necessary for their execution. The proposed architecture has two types of the configuration, *high-level configuration* and *low-level configuration*, according to the level of layers. With *high-level configuration*, a task is planned by connecting *process modules* in series or parallel with respect to one task. The process module encapsulates one process, fault recovery logics, and all necessary information. An internal process is decomposed into the network of behaviors with *low-level configuration*. The configuration is implemented by means of the Petri-net based *configuration diagram*. Petri nets are advantageous for the design of configurations for several reasons [7].

The newly added cleaning task is designed to be performed as follows. First, the workspace is split into several sections. Then, each section is swept by the process *PrCleaning*, and move to the nearest uncleaned section by the process *PrAutoMove*, which is our fundamental navigation strategy. These two processes iterate alternately until the whole workspace is covered.

The process *PrCleaning* is a robust full coverage algorithm using a wall-following technique. The process *PrAutoMove* navigates to the desired position with shortest collision-free path considering several kinds of errors frequently occurred in real applications. The process *PrAutoMove* is already implemented for the former transportation task, the process *PrCleaning* is newly developed.

Fig.2 and Table 1 shows the Petri net model of the process module of *Prcleaning* implemented in *high-level configuration*. If the process ends up successfully, the token is assigned to the P_3 in Fig.2, the next process module starts. If the process fails, the planner reorganizes the existing list of process modules. In the model of *PrCleaning*, the localization fault is considered. If the robot loses its way, the full coverage of the selected area cannot be guaranteed. Therefore, if it occurs, the planner inserts the global localization process in order to find out where it is.

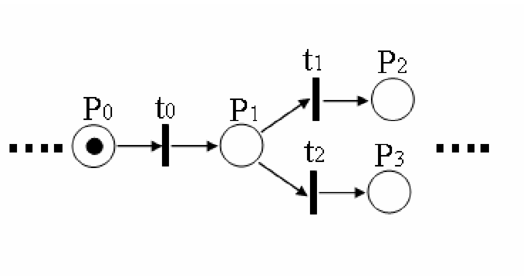

Fig.2. Petri net model of process module for *PrCleaning*

Table 1 Description of places and transition of Fig. 2

	Description
P_0	Standby
P_1	Executing the process *PrCleaning*
P_2	Completing the process *PrCleaning*
P_3	Fault: The full coverage is failed due to localization fault
t_0	*Planner* starts the process *PrAutomove*
t_1	*Planner* completes the process *PrAutomove*
t_2	*Process supervisor (PS)* posts fault message "localization fault"

Fig.3 and Table 2 describes the Petri net model of the process *PrCleaning* in low-level configuration. Since the RSR systems processes two types of wheel mechanisms, cleaning behaviors for both are implemented. Thus, one of the *BhCleaning* and *BhCleaningTwoWheel* is activated according to the parameter encapsulated in the process. If the error event is transmitted from the localizer, the robot stops moving until the success sign is triggered. If the error state is not temporary, the token is moved to the fault place. Then the failure of the process is reported to the planner. The Petri net models and descriptions of the *PrAutoMove* for the *high-level* and *low-level configurations* are shown in [7].

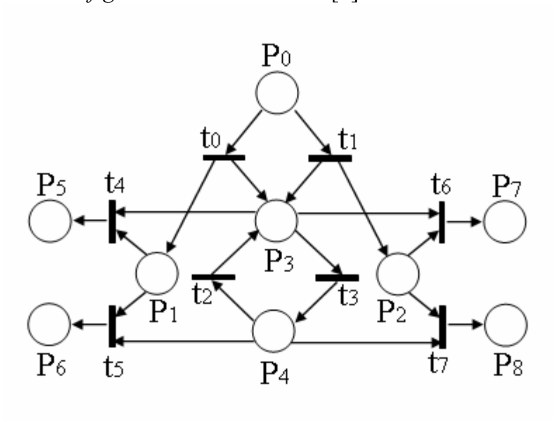

Fig.3. Petri net model of process *PrCleaning*

Table 2 Description of places and transition of Fig. 3

	Description
P_0	Idle (Standby)
P_1 (P_2)	Execute behavior *BhCleaning* (*BhCleaning-TwoWheel*)
P_3	State: Normal localization (Localizer updates a robot position periodically.)
P_4	State: Abnormal localization (Localizer doesn't updates a robot position, and robot navigates only by using odometry)
P_5 (P_7)	Completion of given Process
P_6	Fault: behavior *BhCleaning* is failed due to localization fault
P_8	Fault: behavior *BhCleaningTwoWheel* is failed due to localization fault
t_0 (t_1)	Start behavior *BhCleaning* (*BhCleaning-TwoWheel*)
t_2	Localizer finds out the estimated position is accord with the actual robot position.
t_3	Localizer finds out the estimated position is not accord with the actual robot position.
t_4 (t_6)	Behavior *BhCleaning* (*BhCleaningTwoWheel*) completed successfully
t_5 (t_7)	*PS* terminates the behavior *BhCleaning* (*BhCleaningTwoWheel*) due to the failure of localization

3.2 Component arrangement using *Layered functionality diagram*

The *layered functionality diagram* is a conceptual diagram for arranging various software modules and functions. As shown in Fig.4, The *layered functionality diagram* is composed of three layers, which are deliberate layer, sequencing layer and reactive layer, based on hybrid approach.

The deliberate layer has the function of interfacing with a user and executing planning process. The reactive layer consists of hardware-related modules for sensors and actuators and behaviors, the control command generator in real-time. The sequencing layer is charge of execution of the internal process sent from the deliberate layer by supervising the components in the reactive layer. It also has the algorithms assists the planning process or extracts highly advanced information from raw sensor data.

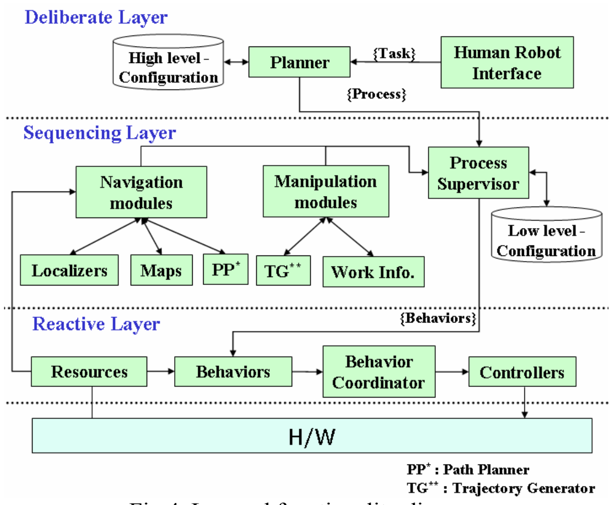

Fig.4. Layered functionality diagram

The *planner*, the *process supervisor*, *configurations*, and the *behavior coordinator* in Fig.4 are core components that are used to build the basis for our architecture. Therefore, it is not able to add or remove these modules. The other parts can be added, removed, or replaced for better performance.

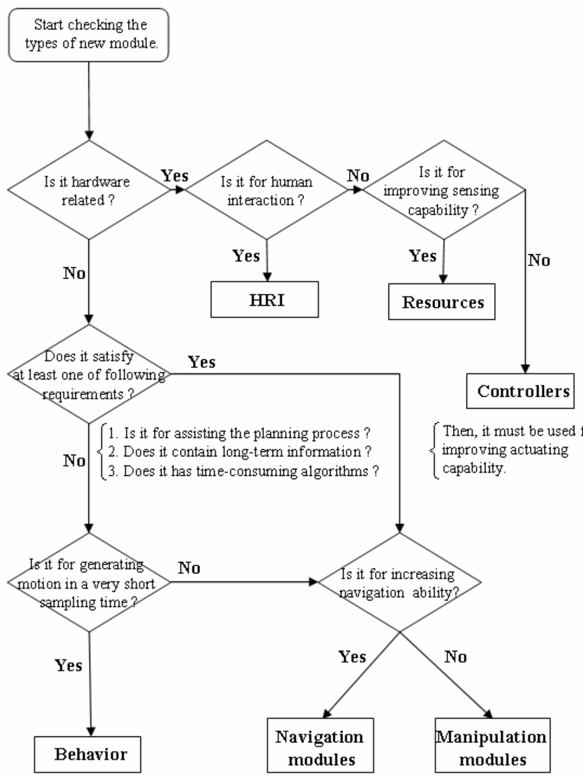
Start checking the types of new module.
Is it hardware related ?
Yes
Is it for human interaction ?
No
Is it for improving sensing capability ?
No
Yes
HRI
Yes
Resources
No
Does it satisfy at least one of following requirements ?
Yes
Controllers
1. Is it for assisting the planning process ?
2. Does it contain long-term information ?
3. Does it has time-consuming algorithms ?
Then, it must be used for improving actuating capability.
No
Is it for generating motion in a very short sampling time ?
No
Is it for increasing navigation ability?
Yes
Yes
No
Behavior
Navigation modules
Manipulation modules

Fig.5. Guideline for arrangement of modules in Layered functionality diagram

When a newly developed module is integrated in our architecture, it should be determined the location in the *layered functionality diagram* at first. Fig. 5 shows the guideline for this arrangement problem. Then, the input and output relations with other modules should be defined. It means we specify which types of information is necessary and where and how to get the information.

In order to implement the cleaning task the *section splitter* and two cleaning behaviors, *BhCleaning* and *BhCleaningTwoWheel* are newly developed. The *section splitter* provides the *planner* with the information about how to generate the sequence of the process *PrCleaning*, and *PrAutoMove*. According to Fig.5, it can be easily found out the *section splitter* should be placed in the *navigation module*. Also, *BhCleaning* and *BhCleaningTwoWheel* are sorted as the *behavior* naturally.

After placing new modules in appropriate positions, input and output relations between new modules and the other one should be defined.

The *section splitter* takes charge of dividing the workspace into several sections if it is a too large open space. The sectioning is more favorable since the performances of sensing and localization fall off in a large open space. The *section splitter* loads the local grid map of the workspace from *maps* firstly. The local grid map represents environments by evenly-spaced grid cells. The division criterion is the existence of isolated obstacles, which is not connected to the wall but located in the middle of free space. The *section splitter* scans the matrix of the local map row by row. If it detects the grid value representing the obstacle in a row, it saves the number of row. Then, it searches the number of row which has no obstacle gird. The *section splitter* set the virtual wall in the middle of them along the row.

After completing the scanning process of whole workspace, the *section splitter* generates the section set, and reports it with the current position gotten from *the localizers*. Then, the planner with *high-level configuration* generates process lists based on this section set. Fig. 6 shows the input and output relations between the *section splitter* and other modules

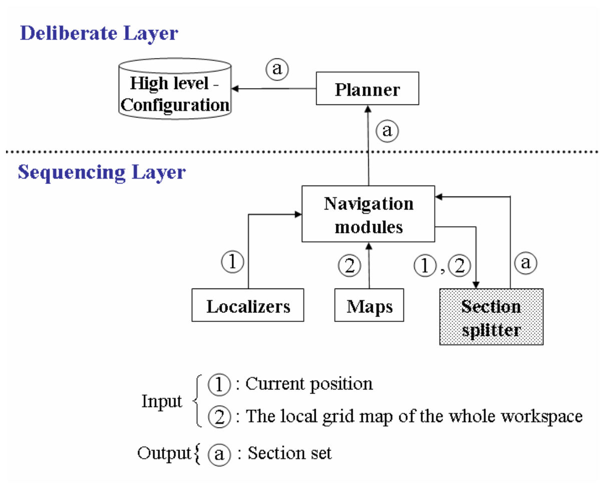

Fig.6. Input and output relations between the *section splitter* and other modules

The developed cleaning behaviors, *BhCleaning* and *BhCleaningTwoWheel*, have same coverage algorithm which is based on the wall-following method. The robot sets the virtual balloon, and moves with keeping it tangent to the wall. These methods guarantees the robot smoothly moves keeping fixed distance with the wall. If the robot makes a round, the radius of the virtual wall is increased. Thus, the robot can fully covers the assigned workspace by repeating this step. Our algorithm is developed in order to minimize the number of times of turning and total moving distance. If the robot turns too often, the cleaning time grows longer due to frequent occurrence of acceleration and deacceleration. The total moving distance should be also reduced in order to decrease the cleaning time and energy consumption. Through the thorough simulations, it is proved that our approach is superior to other existing algorithms like distance transform and plowing method. Another advantage of our algorithm is robust ness with respect to the odometry error, since the algorithm generate velocity commands based on raw sensing data in every sampling time.

As shown in Fig. 7, the behavior initially opens the part of local grid map of section assigned to a single process. In some case, the behavior also takes some process-related parameters such as the position of the virtual wall from the *process supervisor*. In every sampling time, the robot receives the current position from the localizers and raw laser data from the laser resources, and sends Mobile commands to the *behavior coordinator*. When the robot performs cleaning by navigation, the behavior records the cleaned position on the loaded grid map. After the assigned workspace is fully covered, the behavior reports the completion of the given job.

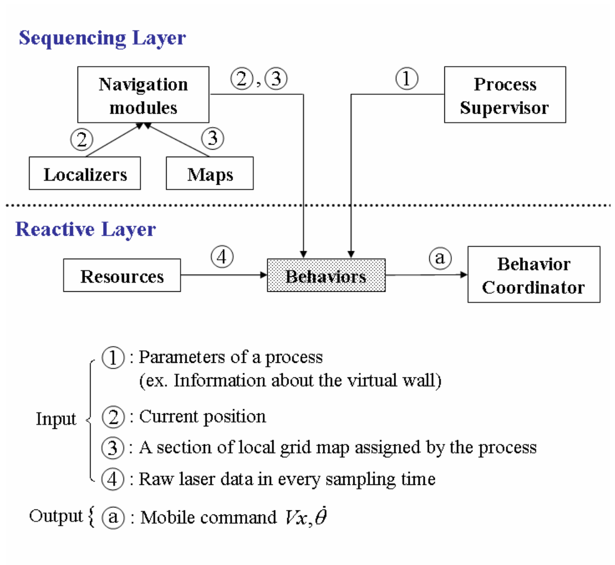
Sequencing Layer
Navigation modules
②,③
①
Process Supervisor
②
③
Localizers
Maps
Reactive Layer
Resources
④
Behaviors
ⓐ
Behavior Coordinator
Input
① : Parameters of a process (ex. Information about the virtual wall)
② : Current position
③ : A section of local grid map assigned by the process
④ : Raw laser data in every sampling time
Output
ⓐ : Mobile command $Vx, \dot{\theta}$

Fig.7. Input and output relations between the cleaning behavior and other modules

3.3 Implementation using *Class diagram*

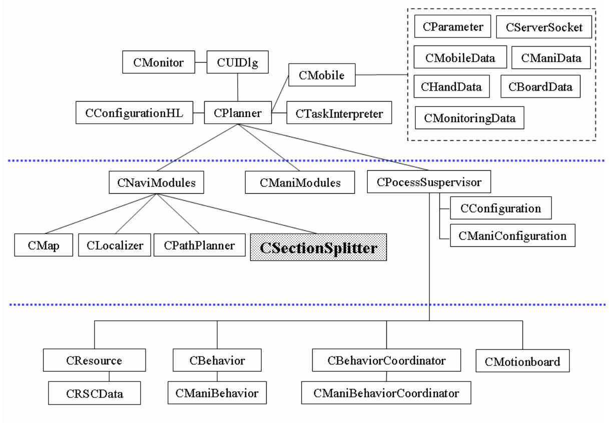

Fig.8. Class diagram

The *class diagram* is designed in order to implement various types of hardware, software, and functional modules for modularity and reusability. It represents instantiation and hierarchy relation between components.

The *class diagram* is defined with two practical reasons. First, the structure of the PSR control program is so complex that it contains about 500 classes. Second, it is necessary that several developers can easily modify or add their components without affecting other modules. Thus, *the class diagram* is designed to makes it easy to describe the structure of software architecture understandably.

The software architecture is coded in C++ using an object-oriented approach. Class diagram resembles the *layered functionality diagram* since each component of the *layered functionality diagram* is implemented as a class. The *section splitter* is encapsulised in a *CSectionSplitter*. It is initiated by *CNaviModules* and interface with other modules through *CNaviModules*.

Class diagram also specifies class inheritance hierarchy for reusability and consistency. For example, all behaviors designed in our architecture should be inherited from *CBehavior*. It defines all common properties and requirements of behaviors, thus developers can make a new behavior through simple modification of some derived functions.

4. IMPLEMENTING TRIPODAL CONTROL ARCHITECTURE INTO A NEW GUIDE ROBOT PLATFORM

4.1 Hardware-dependent modules

In this section, we describe the reusability and scaleability of our architecture by introducing the Jinny system, the newly-built guide robot. The change of mobile hardware configuration is remarkable in Jinny system. The PSRs has holonomic omni-directional mechanisms, but on the other hand, the Jinny is equipped with a two wheel differential mobile base. Moreover, cheaper servo controller is adopted for reducing fabricating cost.

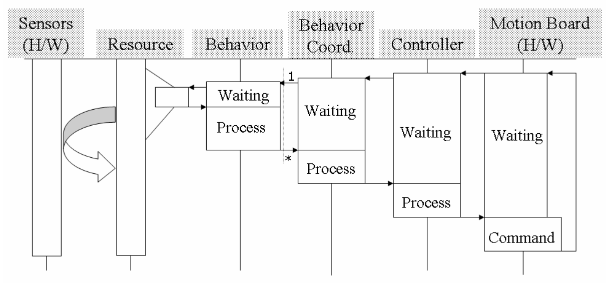

Fig.9. Data flow in reactive layer

Fig.9 shows the structure of the reactive layer in our architecture. Initially, the *behaviors* generate the control command by using raw sensor data from the *Resources* or refined data from the *navigation module*. The *Resources* provide drivers for different kinds of sensor hardware and framework for efficient fusion of theses data. The *resource* is the start point of information flow. Then, the *behavior coordinator* fuses the control commands from one or more *behaviors* and sends them to the *controller*. The *controller* transmits the commands to the hardware motion board every sampling time. This control loop performs very quickly in real time. Several control loops can exist according to the number of the controllers in a single system.

As the Jinny system is developed, because of the Generality and scaleability of this structure of the reactive layer, only hardware-related modules such as the *controller* has changed, but most of them developed for the PSR-1 and the PSR-2 are reused directly. Additionally, sampling time of the control loop, which is closely dependent on the hardware properties, is changed for the new system.

4.2 Human Robot Interface (HRI)

Since the new target system is the guide robot Jinny, we place more weight on the *HRI*. The *HRI* interfaces with a user by receiving a task command and displaying the internal states of a robot.

The inputs to the *HRI* are classified into two types, a reaction input and a command input. The reaction input means the user's request which can be archived by the *HRI* alone. It includes simple dialogs such as "Introduce yourself." or "How's weather today?" The command input is the user's order which should be performed by the planning process. It means the command input can not be completed without helps of other components and algorithms.

To handle the reaction input, the *HRI* should have its own intelligence. The *Robot Knowledge Management System (RKMS)* undertakes this role. It specifies the connection between reaction inputs and proper response scenarios. That is, it extracts some key words from the inputs through receptive elements in Table 3. Then, it searches the results with the highest similarity in its knowledge base. It also applies to the recognition of natural language.

The *RKMS* has a user-friendly GUI interface and uses widely-known web authoring language XML so that developers can easily extend robot's knowledge. Moreover, it can be administrated in a remote site since it is implemented as a web server.

Table 3 describes the interaction element of the *HRI* in Jinny system.

Table 3 The interaction elements of the *HRI* in Jinny system

Receptive elements	- Voice recognition
	- Touch screen
	- 12 LED buttons
Expressive elements	- Voice synthesizer
	- Screen (displays animated agents, pictures, and videos)
	- Gestures (using mobile base, 2-DOF neck system, and two 1-DOF arm system)

5. EXPERIMENTS AND RESULTS

5.1 Cleaning experiments

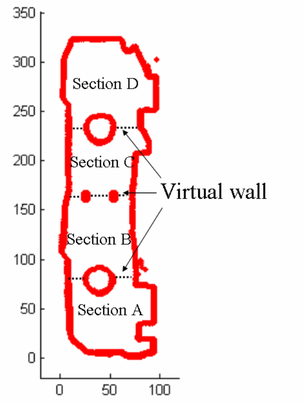

Fig.10. The map of a workspace for cleaning experiments

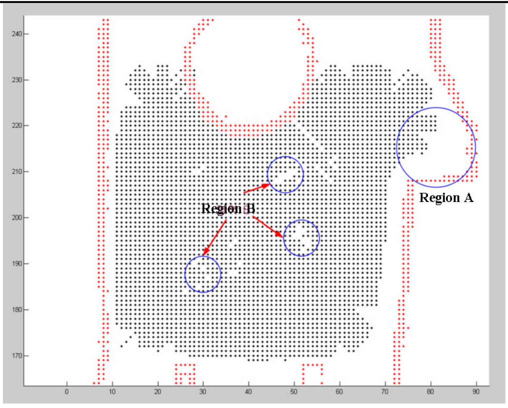

Fig.11. The results of coverage of Section D

The developed cleaning task is implemented and tested on the PSR-2 platform. Fig. 10 shows the map of a workspace whose size is about 10m × 32m. As described before, the workspace is split into four sections and virtual walls are set in order to perform cleaning by wall following algorithm. Therefore, four times of *PrCleaning* and several times of *PrAutoMove* are necessary to complete a cleaning task of the given workspace. The number of execution of *PrAutoMove* and the order of these processes are changed according to the robot's initial position.

The results of coverage of section D is also presented in Fig.11. Fig.12 shows the PSR-2 in cleaning experiments.

Fig.12. PSR-2 in cleaning experiments

5.2 The guide robot Jinny

The Tripodal schematic control architecture is successfully implemented into the new developed guide robot Jinny. The Jinny is demonstrated at the 2003 Korea Science Festival from August 13th to 21st as shown in Fig.13. During this exhibit, the Jinny is tested in a real environment over and over, and analyzed its interaction with visitors.

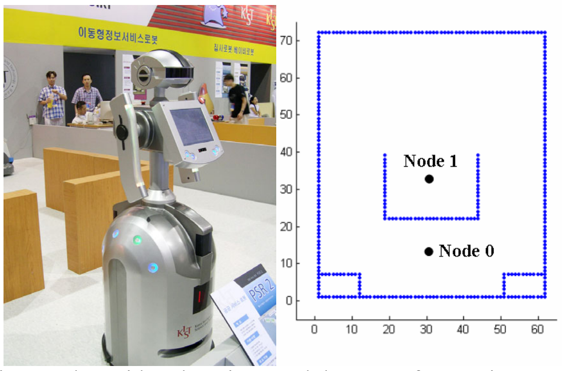

Fig.13. The guide robot Jinny and the map of an environment

Fig.13 shows the map of the environment at Science Festival exhibit hall. The Jinny moves between node 0 and node 1 while introducing itself and explaining other displayed robots. Although there are a couple of persons in the environments, the Jinny can navigate without collision as like the PSR-1 and the PSR-2.

The Jinny performs several interesting service jobs in response to the user's request. For example, it plays a simple game with visitors, and dances to the music. Also, it can provide the information about today's weather and stock quotations to the visitors through the internet.

6. CONCLUSION

This paper addressed our experience about the development of PSR and Jinny systems with the Tripodal schematic control architecture. In this paper, we showed practical advantages of the proposed architecture by giving examples. First, we explained how to add new cleaning task using Tripodal architecture approach step-by-step. Then, we introduced the implementation of the guide robot Jinny for the reusability and scaleability of our architecture.

Though the hardware configurations as well as software algorithms are more complex and more accumulating, the Tripodal schematic control architecture successfully managed it. Experimental results clearly showed that the developed strategy is useful for extending robot's ability and developing new platform.

Although not reported in this paper, some service tasks are also successfully implemented. A backward tracking task with multiple trailers is also accomplished. This task is quite complex job since several many control algorithms are necessary and hardware configurations are quite different.

REFERENCES

[1] Erann Gat, Three-Layer Architectures in D. Kortenkamp et al. eds. Artificial Intelligence and Mobile Robots, AAAI Press, MA; 1998, pp.195.

[2] R. C. Arkin, Behavior-Based Robotics, The MIT Press, MA; 1998, pp.214.

[3] Reid G. Simmons, "Structured Control for Autonomous Robots," IEEE Trans. on Robotics and Automation, vol. 10, no. 1, Feb. 1994, pp. 34-43.

[4] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Dellaert, and Dieter Fox, "MINERVA : A Second-Generation Museum Tour-Guide Robot," in Proceeding of the IEEE Conference on Robotics and Automation, Detroit, Michigan, USA, pp. 1999-2005, 1999.

[5] Fabrice R. Noreils and Raja G. Chatila, "Plan Execution Monitoring and Control Architecture for Mobile Robots," IEEE Trans. on Robotics and Automation, vol. 11, no. 2, Apr. 1995, pp. 255-266.

[6] Mattias Lindstrom, Anders Oreback, and Henrik I. Christensen, "BERRA : A Research Architecture for Service Robots," in Proceeding of the IEEE Conference on Robotics and Automation, San Francisco, CA, USA, pp. 3278-3283, 2000.

[7] Gunhee Kim, Woojin Chung, Munsang Kim, Chongwon Lee, ¨Tripodal Schematic Design of the Control Architecture for the Service Robot PSR,〃 in Proceeding of the IEEE Conference on Robotics and Automation, Taipei, Taiwan, 2002.