2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph

Joaquín L. Fernández, *Member, IEEE*, Rafael Sanz, Enrique Paz and Carlos Alonso

Abstract— **Most of the mobile robot control frameworks are based on a middleware layer with several independent modules that implement primitive actions and report events about their state. These modules are usually connected with different inter-process communication mechanisms. Here, we propose to use hierarchical interpreted binary Petri nets to coordinate the activity of these modules. Tasks are described using an interpreted Petri net editor and saved in a xml file. A dispatcher loads these files and executes the different Petri nets under user requests. A monitor that shows the state of all the running nets is very useful for debugging and tracing purposes. The whole system has been applied to a guide robot (vixiabot) that has been guiding users in a public event (XuventudeGalicia.net) for three days in April 2007 and is now extended to a multirobot surveillance application.**

I. Introduction

BUILDING mobile robot applications that work for long periods of time in a real environment is a difficult task. Several research groups have adopted different control frameworks solutions [1][2][3]. However, only a very few of these projects can finally work without human supervision for a long period of time mainly because of their complexity. Nevertheless, many simple service and industrial applications can benefit from the use of autonomous robots for executing typically repetitive tasks performed nowadays by humans.

In order to build robust mobile robot applications, some of the solutions that have been successfully used in complex industrial manufacturing applications might be adapted. For example, programs to control commercial industrial manipulators are build from two different kind of basic commands: First, a set of basic application independent movement functions such as move a manipulator tool to a defined point in the workspace describing a defined trajectory and, second, a set of application specific functions such as activate/deactivate terminator element (grab/release, start/stop painting, etc). Regarding the mobile robots applications, similar sets of commands can be defined: First, a set of general basic application independent navigation functions such as move to a defined point in the environment and, second, a set of application specific functions such as "say text", "grasp object", etc.

This work was supported in part by the Spanish Comisión Interministerial de Ciencia y Tecnología, CICYT, project DPI2005-06210. The. Authors are with the Dep. Ingeniería de Sistemas y automática, University of Vigo, 36200 Vigo, Spain (phone: +34 986 812222; fax: +34986 814014; e-mail: joaquin@uvigo.es, rsanz@uvigo.es, epaz@uvigo.es).

A modular layered approach has proven to be a successful paradigm in current mobile robot service applications. The majority of control systems for autonomous mobile robots are characterized by a three layer architecture [1][2][3]. The planning layer obtains plans to achieve high-level goals, the executive layer sequences and monitors task execution and, finally, the functional (middleware) layer provides the basic computational threads for robot action and perception.

Whereas some details about functionality and names of each layer are still matter of discussion [4] [5], the functional layer has a uniform structure in all architectures: a set of components, communicating with each other and with the upper layers. Some of these components provide access to sensor data, others send commands to actuators and, finally, the last set provides different basic functionalities (localization, path planning, obstacle avoidance, etc.) or behaviors (follow wall, etc.). Upper layers send request and receive information about successful termination or failure together with computation results.

Returning to the manipulators analogy, the functional layer implements the basic general functions, the executive layer should be the "program" in charge of sequencing the activity of the functional layer components and the planning layer functionality is mainly done by the programmer.

Petri net based formulation of tasks for industrial robots and other dynamic systems [14] [15] exists, especially for manufacturing tasks. The main industrial automation companies use IEC 61131-3 compliant programming environments that include the Sequential Function Chart (SFC), language that can be considered a special case of binary interpreted Petri nets.

In this paper we propose an implementation of the executive layer based on Petri nets. Petri nets are a powerful tool to model, design and analyze distributed, sequential and concurrent systems [6]. This provides several advantages building mobile robot applications:

1) *Module reusability*. Navigation modules and even some application specific modules will remain unchanged from application to application.
2) *Reduce development time*. In different projects of the same kind of applications, most of the modules remain without changes and only the edition of the Petri nets will be necessary to personalize the application. For example, different robot museum projects can be built with the same modules changing in the Petri nets the places to visit, the texts to say and some other minor changes.

3) *Facilitate maintenance*. Tracing and debugging problems are easier to settle when the system state can be seen looking at the evolution of a Petri net rather than monitoring a set of variables.
4) *Training*. Almost everybody that has worked or learn to use IEC 61131-3 compliant programming environments (Siemens S7 Graph, Graphcet, etc.) will be able to program with our tool *RoboGraph*.
5) *Analysis and test*. Petri net properties also make them good candidates for qualitative (un-timed models) performance evaluation and quantitative (timed models) performance evaluation of robotic tasks. Significant research has been done in this area for industrial applications [6][15] and also some in mobile robots tasks [7][9].

The rest of this paper is organized as follows. Next section introduces the works related to this research. The control architecture used in our mobile robots is presented in section III. After describing the Petri net plan representation in section IV, the *RoboGraph* implementation details are shown in section V, while the extension of this framework to multirobot applications is specified in section VI. Finally, section VII concludes the paper.

II. RELATED WORKS

Within the robotics and automation areas, Petri nets have been widely used to model and study flexible manufacturing systems (FMS) [6][15], where the major applications are the modeling, performance analysis and scheduling. Several works in this field have already been mentioned in the introduction.

Some researches have already point out that several plan representations used in mobile robots can be mapped to Petri nets. For example, in [10] a plan representation using partially ordered plans (POPs) [11] that include operators as in STRIPS [12], ordered constrains and conditional actions is converted to a Petri net model. In [16] a methodology for automatically transforming conventional task-variable graphs representing the execution levels of intelligent control architectures in Petri nets is presented. Finally, in [13] a Robotic Task Model (RTM) based on Petri nets is introduced for a distributed robotic system.

Some applications based on mobile robots [7][9] use Petri nets to model and evaluate plans at different levels that go from motion control [19] to task supervision [14]. A few papers have also reported the use of Petri nets in mobile robots for coordination, hardware resource handling and planning [17] combining some AI planning system with Petri nets.

Most of the works mentioned above use Petri nets to modeling, analyze and even testing [18] the control of mobile robots. We propose to extend the use of Petri nets as a high level task application programming language through an IDE, in a similar way industrial automation companies use IEC 61131-3 compliant programming environments. This way, besides defining the Petri nets to model, analyze and test, they will also be used by a dispatcher to control the different functional modules of the control architecture. For that purpose, a Petri net programming environment named *RoboGraph* has been defined. *RoboGraph* provides also some very useful visual debugging tools that can show in real time or reproduce the evolution of the status of the different tasks through the evolution of the marks on the associated Petri nets.

III. CONTROL ARCHITECTURE

ISANAV is a modular control architecture is organized as shown in figure 1. Even though the different modules are structured in four sets, they can be mapped in the three layer architecture popularized by Bonasso de al. [1]. The hardware servers and control set implement the functional layer while *RoboGraph* dispatch implements the executive and planning layer. Finally, ISANAV includes a set of processes to interact with the users and to connect to other process for multirobot applications.

The navigation platform is based on CARMEN [8] and some modules, such as localize, navigator and base hardware servers remain basically the same. Unlike CARMEN, motion control is divided into high-level (strategic) planning [20] and lower-level (tactical) collision avoidance using the Beam method [21]. CARMEN integrates obstacles in the map and plans a new trajectory in order to avoid obstacles. Integrating all but the lowest-level motor control into a single module can produce optimal plans. However, due to the lack of precision in the localization system, the obstacle integration process can narrow some openings in the map. When the opening is only a little bit wider than the robot diameter, this difference can lead the path planning to discard a possible path through that opening. We have ob-

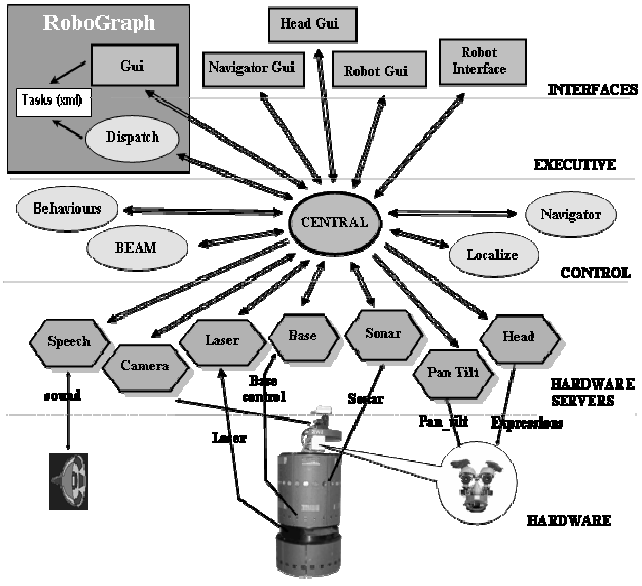

Fig. 1. ISANAV Control architecture. Different modules are divided in several sets. The hardware servers set reads sensor data and controls actuators. The control set provides several basic functions that can be used by other modules of this set and the executive set.

served this behavior in several points of our office environment using CARMEN, since the robot has to go through very narrow doors.The hardware server modules handle hardware interaction, providing an abstract set of actuator and sensor interfaces and isolating the control methods from the hardware details. Most of the hardware devices are connected to a CAN bus using RoboCAN [22]. Some of these devices are used in navigation, such as the laser and sonar while others are specific for the application, such as the robot head, sound and speech system, etc. The hardware servers also provide low-level control loops for rotation and translation velocities. Thanks to this layer, changes in hardware components can be made without modifying higher layers modules while keeping the same interface.

The control modules integrate sensor and motion information to provide improved sensor odometry, offering basic navigation capabilities (localization, path planning, follow path, etc) and basic application specific functions (say text, make expression, etc.).

Robograph dispatch coordinates the execution of the control modules and sequences their functions according to the plan defined as a Petri net. This module is described in detail in the following sections.

There are several interfaces modules to allow the user to interact with the application, debugging control modules and exchange information with a web server. Users can also connect via web, monitor and interact with the robot.

Each module in figure 1 is a Linux process that exchanges information with other modules using IPC Inter Process Communication [23]. Developed at Carnegie Mellon's Robotics Institute. IPC provides a publication-subscription model. Each application connects with the central server, and specifies what types of messages it publishes and what types it listens for. Any message that is passed to the central server is immediately copied to all other processes subscribed.

The process of building a mobile robot application using this framework includes programming on different levels. First, hardware server and control modules need to be implemented. Modules that execute navigation tasks can be used from one application to another. At the next level, the executive layer, it is necessary to build a module or sequencer that sends requests and receives the outcomes of the functional layer modules. This module usually varies from one application to another.

IV. PETRI NET PLAN REPRESENTATION

Petri nets have been widely used to model, design, execute and evaluate tasks in manufacturing dynamic systems. As we have seen in the related work section, some researches have also point out that several plan representations used in mobile robots can be mapped to Petri nets. In this work we use hierarchical binary interpreted Petri nets.

As a simple example, figure 2 shows the Petri nets that can be used for the task "GO POINT". The Petri net on the left is the simplest implementations. There is only one initial mark in the place labeled "Set Goal", while the "END" place has been selected as a final place (red). The task ends when only the final places are marked or there are no marks on the Petri net if no final places have been defined.

Action publish message "set_goal" is assigned to "Set Goal" place. The parameters for this message are the goal coordinates. Module Navigator (figure 1) is subscribed to this message. Every time some module publishes it, the navigator will get a copy, plans the path and publishes another message "navigator_plan" with the planned path.

Transition labeled "Path" has associated the event receive "navigator_plan" message. When this event occurs and "Set Goal" place is marked, the transition will be fired.

The place "Go" has the action publish message "follow_path" assigned. The BEAM module will handle this message, activating the "follow path" behavior. Once the goal is reached, the "autonomous_stopped" message is issued. This message is associated to transition "STOP". Firing this transition will remove the mark from "Go" place and set a mark on "END". The "END" place is the only final place, therefore, the task will finish publishing the corresponding "end_task" message.

Since the robot is working on a real complex dynamic environment, several problems can arise while following the path. A Petri net that takes into account some of these problems is shown on the right part of figure 2. While following the path (place "Go"), the navigation system can publish a message "autonomous_stopped", but with a parameter setting that the goal has not been reached. This event is associated to transition "No Goal". Another mechanism to detect possible problems is the use of a time out with the maximum time needed to finish the action follow path. This is done here using a timer that can be initialized in the transition "Path" and validated in transition

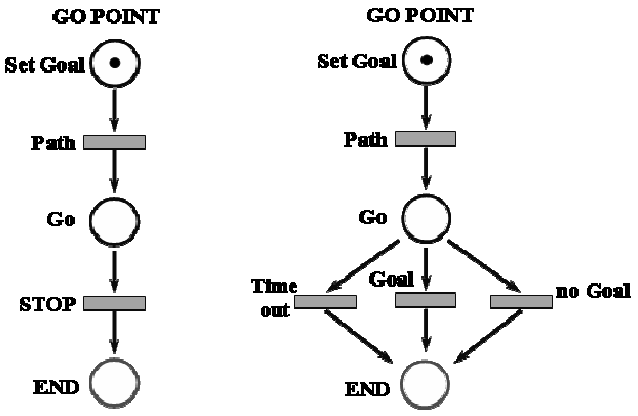

Fig. 2. Two different ways to implement the task "GO POINT" with Petri nets. The simplest (left) does not tackle any incidence on the follow path action. The one on the right show some mechanisms that can be used to take into account for problems on the execution of some actions (Go).

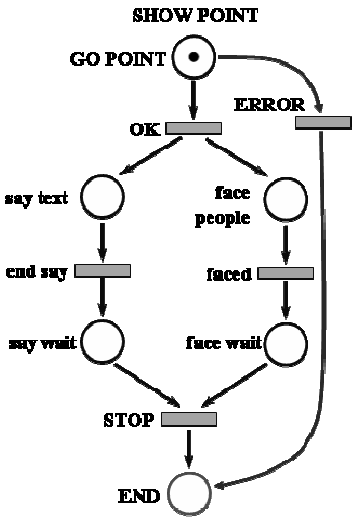

Fig. 3. Petri net that implements the task "SHOW POINT". Initial place includes the execution of task defined in figure 1. When the subnet finishes it execution a message is issued and a string return value is used to know the outcome of the execution.

"Time out". While the "Go" place is marked any of the three transitions can be fired. For each of the three transitions there is an action that assigns a different value to the global string variable "return" that will be sent as a parameter with the message "end_task" when the Petri net (task) finishes. This parameter is used in the Petri net of figure 3 by transitions OK and ERROR to check if the "GO POINT" Petri net has reached the goal.

Figure 3 defines "SHOW POINT" task that uses the task "GO_POINT" defined above. In the initial place a "Run_Petri_Net" message is issued with several parameters, such as the name of the Petri net (GO_POINT), Petri net identity PN1, user identity (by default the same as requested "SHOW POINT"), running mode and goal coordinates. *RoboGraph* dispath is subscribed to this message as well as the "End_Petri_Net" message. If the "End_Petri_Net" message form PN1 is received reporting success in the "GO_POINT" task, transition "OK" will be fired and actions associated to "say text" and "face people" are executed in parallel. Both threads of execution are synchronized in transition STOP because places "say wait" and "face wait" need to be marked for the transition "STOP" be fired.

Resource sharing, scheduling, and any other of the nice features of Petri nets can be used to define almost any complex task.

V. RoboGraph

Figure 1 shows the programs that form *RoboGraph*. The GUI (Graphical User Interface) is a development tool that make possible to create, edit and monitor the execution of the different tasks while the DISPATCH is in charge of executing those tasks.

A. GUI

This program can work in three different modes and it is possible to switch at any time from one to another: Editor, Monitor and Play Logger.

In editor mode, the user can create new tasks using a simple and intuitive Petri net graphical editor. Once the Petri net is constructed by selecting and dragging different elements (places, transitions, arcs and marks), actions, associated to places and transitions, and conditions, associated to transitions, must be defined for the interpreted Petri net.

Actions can be commands implemented in any module in the control architecture of figure 1. These commands can be selected from a list automatically generated by the GUI. Each command is an IPC message and the user must define the command parameters that will automatically appear in a new window when that command is selected in the editor.

When Dispatch executes the Petri net, the IPC messages assigned to places and transitions will be published as the net progress.

Conditions can be events produced by any module in figure 1 that can be selected from a list generated automatically by the GUI. An event can be the arrival of an IPC message, a condition on some IPC message parameter or any logical expression on several parameters over the same or different IPC messages.

Timers are a tool widely used in automation that comes very handy here. In addition, in our applications we have also used them as an error detection mechanism in order to time some actions of different modules in case they get stuck. Actions can start a timer while conditions can test the value of a timer.

Global variables are used to get starting data and store information to share conditions and events in different places and/or transitions.

GUI in monitor mode connects to central (IPC) and subscribes to different dispatch messages that show the status of the different running or waiting Petri nets. Every running Petri nets is shown in a different tab with the actual marking. When dispatch evolves a Petri net marking, an IPC message is issued and GUI will update the monitor tabs. Therefore, using monitor we can see in a snapshot the status of the system, since the marking of the running Petri nets represents their status. This is a very helpful tool when debugging an application. An information window with the queued tasks (Petri nets) can also be displayed.

Mobile robots operating in the real world can fail for many reasons because it is almost impossible for the programmer to predict all the circumstances that might be encountered. In order to be able to trace different problems, an XML log file with Dispatch IPC messages is created in running time. The system administrator can then run GUI in play-logger mode, open the log file and play it at the same pace as in the real execution. Different tabs with the running Petri nets will be shown as in monitor mode. Besides the regular play option, the user can monitor the log file step by step, jump to a defined place in "execution" as many commercial programming development environments for

high level languages (C, Java, C++, etc.). Finally, the user can see different details about the IPC messages.

B. Dispatch

Dispatch schedules the different actions of the functional (basic actions), executive (other Petri nets) and interface layer (user and web interfaces), as well as the synchronization with the events produced. The interaction with other modules in the architecture is performed by publishing and subscribing to messages. This way, problems in a module, such as a deadlock problem, do not block dispatch and we can set up simple mechanisms to detect and recover from a failure or exception situation.

When starting, dispatch subscribes to the task execution or cancellation requests. Every requested message has an owner, priority and execution mode (serial or parallel). Priority and owner define the execution priority and the ability to kill or stop a task. Execution requests that cannot be executed at the reception time, they are stored in different queues according to their priority. A task execution request can come from different modules (figure 1), such as user interface modules (user requests) or even from the *RoboGraph* dispatch when the task is an action associated to some running Petri net.

The execution of a new task starts loading the interpreted Petri net from a XML file, setting the initial marking, subscribing to all the IPC messages referenced in the events and executing the actions associated to the marked places. The Petri net can only progress with the arrival of IPC messages or the end of a timer.

Each time a change in the status of a Petri net (start, stop, evolve) or in the waiting queues (new requests added or removed) is produced, a new IPC message reporting that change is issued for GUI monitor mode and stored in the log file for GUI play-logger mode.

VI. MULTIROBOT ARCHITECTURE

We have used the architecture described in figure 1 in a mobile robotic guide that was working during three days in the "Palacio de Congresos y Exposiciones de Galicia", Santiago de Compostela (Spain) for the "Xunventude Galicia Net" event. For this application, a modified Peoplebot model from Activemedia shown in figure 4 was used.

In this kind of applications the environment is a set of stands, most of them mounted a few days before. Furthermore, some of the tasks are not fully defined until a few hours before the starting of the event. Therefore, a tool like *RoboGraph* to quickly create, change and debug tasks becomes necessary.

Our current research addresses architectures that extend the approach described in figure 1 to control several mobile robots (figure 5). This work is currently being developed as a part of a research project. The project falls within the context of surveillance systems using mobile robots. More specifically, it is based on a set of robots guided in dynamic,

Fig. 4. Robot Vixiabot is giving a tour in "Palacio de Congresos y Exposiciones de Galicia", Santiago de Compostela (Spain) for the xuventudeGalicia.net event in April 2007.

non-fully specified environments and working under real-time restrictions. The robots must be able to carry out plans remotely and react to unexpected events.

Figure 5 shows the proposed architecture for two robots connected to Ethernet, but it is extensible to multiple robot systems. Here *RoboGraph* is been applied in two different levels. The first one is to control each robot as it has been described in last sections. The second, at the highest level, is used to coordinate the work of different robots.

The communication system, named JIPC, connects different robots, *RoboGraph* and the graphical user interfaces. Graphical user interfaces can be running on the robot or a java applet on a web Page for web based applications. JIPC implements a publish-subscribe policy similar to IPC. However, a few changes have been made in order to be able to select the receiver when publishing mes-

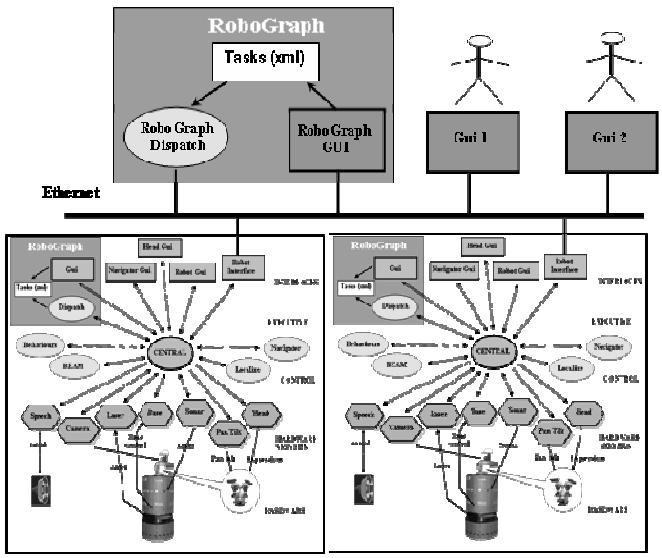

Fig. 5. Multirobot control framework. RoboGraph can be used to program multirobot applications.

-sages, to define an access control mechanism and to remove some native libraries to start the graphical interface from any web browser. Different robots have the same control frameworks and identical messages to start same actions. IPC does not allow discerning the sender when subscribing to messages or the receiver when publishing a message. The web interface module maps IPC messages to JIPC messages. The main difference of this new *RoboGraph* used in multirobot programming is the use of a different communication interface. Even though now it is necessary to define not only the commands but also to know which robot should be sent and who request them, these are fields in the messages and therefore *RoboGraph* does not need to be changed.

VII. CONCLUSION

Through the use of a Petri net based programming environment to implement the executive layer of a control mobile robot framework we have shown that it is possible to build and maintain robust mobile robot applications in a quite fast and intuitive way. The programming environment, called *RoboGraph,* has been designed to have similar functionalities to other 61131-3 compliant environments built for automation companies to program industrial applications. We expect to increase its functionality by adding a Petri net formal verification function.

RoboGraph can be adapted to be used in any mobile robot control framework that includes a functional layer with a set of components that provide access to sensor data, send commands to actuators and different basic functionalities (localization, path planning, obstacle avoidance, etc.) or behaviors (follow wall, etc.). However, the use of frameworks where the modules are implemented in an independent way (different processes) communicating with each other and with the upper layers via some inter-process communication mechanism provide several advantages. For example, reusability of modules, robustness since a failure in a module does not implies the whole system to fail, and is also easier to maintain the system.

ACKNOWLEDGMENT

We would like to thank all the people that have influenced this work. In particular to Reid Simmons for his helpful advices about the use of IPC.

REFERENCES

[1] R. Bonasso, D. Kortenkamp, D. Miller and M. Slack, “Experiences with an architecture for intelligent, reactive agents”. Journal of Artificial Intelligence Research Vol 9(1), pp: 237-256, 1997

[2] E. Gat. On three-layer architecture. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, Artificial Intelligence and Mobile Robots, pages 195–210. AAAI Press, Boston, MA, 1998.

[3] R. Simmons, R. Goodwin, K. Haigh, S. Koenig and J. O’Sullivan, “A layered architecture for office delivery robots”. In Proc. First International Conference on Autonomous Agents. 1997.

[4] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. “The CLARAty architecture for robotic autonomy”. In IEEE Aerospace Conference 2001, Big Sky, MT, 2001.

[5] S. Caselli, F. Monica, and M. Reggiani, “YARA: A Software Framework Enhancing Service Robot Dependability”. In Proceedings of the 2005 IEEE International Conference on Robotics and automation. Barcelona, Spain, April 2005

[6] M.C Zhou and K. Venkatesh.”Modeling, simulation, and control of felxible manufacturing systems”. World Scientific, 1999.

[7] F. Y. Wang, K.J. Kyriakopoulos, A. Tsolkas and G. N. Saridis, “A Petri-Net Coordination Model for an Intelligent Mobile Robot”. IEEE Transactions on Systems, man and cybernetics, Vol. 21, No. 4, July/august, 1991.

[8] M. Montemerlo , N. Roy, S. Thrun, “Perspectives on Standardization in Mobile Robot Programming : The Carnegie Mellon Navigation (CARMEN) Toolkit”, Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2003). 27-31 Oct. 2003. Vol. 3, pp: 2436- 2441

[9] G. Kim, W. Chung, M. Kim and C. Lee, “Control architecture Design and Integration of the Autonomous Service Robot PSR”, Proceedings of the 11th International Conference on Computer Applications (ICCAS 2002) pp: 2379-2384.

[10] J. King, R. K. Pretty, and R G. Gosine, “Coordinated Execution of Tasks in a Multiagent Environment”, IEEE Transactions on Systems, Man and Cybernetics –Part A: Systems and Humans, Vol. 33, No. 5, September 2003

[11] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[12] R. E. Fikes and N. J. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem,” Artif. Intell., vol. 59, no. 1–2, pp. 227–232, 1971.

[13] D. Milutinovic and P. Lima, “Petri net models of robotic tasks,” in Proc., IEEE International Conference on Robotics and Automation (ICRA), 2002, pp: 4059 – 4064.

[14] A. Lehmann, R. Mikut, T. Astour, “Petri nets for task supervision in humanoid Robots“, VDI BERICHTE, 2006, VOL 1956, pp: 71-72.

[15] J. Flochova, “A Petri net based supervisory control implementation” in Proc., IEEE International Conference on Systems, Man and Cybernetics, 2003. Volume: 2, On page(s): 1039- 1044

[16] M. Caccia, P. Coletta, G. Bruzzone, G. Veruggio, “Execution control of robotic tasks: a Petri net-based approach”, Control Engineering Practice, 13 (8), p.959-971, Aug 2005.

[17] T. Asfour, D. Ly, K. Regenstein, and R. Dillmann, “Coordinated task execution for humanoid robots,” in Experimental Robotics IX, ser. STAR, Springer Tracts in Advanced Robotics. Springer, 2005.

[18] A. Chandler, A. Heyworth, L. Blair, D. Seward, “Testing Petri Nets for Mobile Robots using Groebner basis”, 21st International Conference on Application and Theory of Petri Nets Aarhus, Denmark, June 26-30, 2000

[19] K. Kobayashi, A. Nakatani, H. Takahashi, and T. Ushio, “Motion planning for humanoid robots using timed Petri net and modular state net,” in Proc., IEEE International Conference on Systems, Man and Cybernetics, 2002.

[20] A.R. Diéguez, J.L. Fernández, R. Sanz, “A global motion planner that learns from experience for autonomous mobile robots”, Robotics & CIM Vol. 23 pp: 544–552 (2007), Ed. Elsevier.

[21] J.L. Fernández, R. Sanz, J.A. Benayas and A.R. Diéguez, “Improving collision avoidance for mobile robots in partially known environments: the beam curvature method.” Robotics and Autonomous Systems. vol. 46, pp. 205-219, April 2004.

[22] Fernández, J.L., Souto, M.J., Losada, D.P., Sanz, R., Paz, E., “Communication framework for sensor-actuator data in mobile robots”, Proceedings of ISIE 2007. IEEE International Symposium on Industrial Electronics, 2007. pp: 1505-1507

[23] R. Simmons, “The interprocess communications system (IPC). http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html. Accessed: 2007-08-29.