

# A Dynamic Self-Reconfigurable Mobile Robot Navigation System

Yan Meng, *Member, IEEE*

Department of Electrical and Computer Engineering  
Stevens Institute of Technology, Hoboken, NJ 07030  
ymengl@stevens.edu

**Abstract** – A mobile robot navigation system must adapt to the highly dynamic environments and emergencies under the real-time constraints. Sometimes some sensors may not work well with new environments while others may need to swap in at runtime to continue the navigation. This paper presents an agent-based embedded system platform which can dynamically reconfigure the robot system on the fly by integrating FPGA hardware and high-performance microprocessors system. Several dynamic reconfiguration models are described to improve the system efficiency with low reconfiguration latency. The reconfiguration architecture specifically for vision system is also presented. This platform can be easily extended to other robot systems, such as server robots, space robots, and multi-robot systems.

## I. INTRODUCTION

Researchers have done extremely works to develop various mobile robotic systems. Navigation is one of most important features of the mobile robots. Almost all the developed mobile robot systems are designed to work efficiently under a specific certain environment, such as structured indoor office building with good illumination condition, or unstructured outdoor sparse uneven terrain during daytime. Very few mobile robots, to our knowledge, can move from one environment to another dramatically changed one without losing some important performance or capabilities. For example, when a robot enter a dark room from outside during daytime, the normal CCD camera might work efficiently in the darkness, and an infrared camera or a night vision gear (NVG) may have to swap in the system to make the robot continue its navigation under new condition.

Another assumption for most robot systems is that all the sensors are working perfectly during their life time. However, in the real-world, some sensors may lose their performance or capabilities either due to the changing environment, as we mentioned above, or due to their own physical malfunction under extremely hazardous environment, such as in a burning building. When such case happens, the robot systems have to adapt themselves to deactivate the faulty sensor units and activate some redundant sensor units if there is any. Otherwise the sensor information would bring huge bias to the system.

As researchers vigorously develop advanced control and processing schemes for mobile robot navigation, there exists a need for a scalable real-time embedded system architecture capable of robust application-specific processing at run time without prohibitive high costs of building another robot if the whole environment of the

navigation changed dynamically. As the quantity and diversity of mobile robot navigation environment and requirements and computationally intensive applications continues to increase, the architecture of real-time embedded system for mobile robot navigation system must respond with greater flexibility, processing capacity, and performance.

Normally a mobile robot navigation system includes several tasks, such as path planning, obstacle avoidance, emergency stop, sensor signal processing, sensor manager for sensor fusion, actuators, and actuator manager. Most robot systems are implementing all these tasks in software. It would be a challenging job for a software engineer to implement all the tasks under real-time constraints especially with computation-intensive image information under dynamic environment. The expensive ASIC can fulfill the speed criteria, however, it is a complicated and expensive procedure if a slight change occurs. Recent attempts to find new ways to speed up computation take advantage of the Field Programmable Gate Arrays (FPGA), since they offer the speed similar to an ASIC with a flexibility equal to or even higher than a general processor.

In order to increase the robot's robustness and flexibility with the real-time constraints, in this paper, we propose a dynamically reconfigurable real-time embedded system platform by integrating high-performance microprocessors with a reconfigurable FPGA to enable a mobile robot system to adapt to dynamic environment on the fly with the minimum loss of its performance and capabilities.

The software/hardware co-design would benefit the system performance significantly by providing the lower latency and higher bandwidth. However, at the same time, it also introduces the hardware/software synchronization issue to the system. If a generic methodology can be applied to both software and hardware during the design phase, it would greatly simplify the system design and improve the performance significantly. Therefore, a multi-agent system paradigm is proposed in this paper as a unified methodology for hardware/software co-design. In our robot system, one agent represents a task instead of a robot. If a task is partitioned in software, it is called software agent, otherwise, it is hardware agent. Since both software and hardware agent share the same agent model, it is possible for them to share the same design methodology. There only exists the implementation difference, that is, software agents are implemented on the RISC processor using C/C++ language while hardware agent is implemented on FPGA using hardware description language (such as VHDL).

Dynamic reconfiguration is based on the idea that parts of the system remain available during the reconfiguration. Although disruption is unavoidable, the impact of the disruption should be minimized, as well as the duration of the effects of this disruption. The reconfiguration mode should introduce minimal overhead during normal operation. Several agent reconfiguration modules are described in this paper. Vision system is a powerful sensor for mobile robot navigation. However, the intensive computation makes it difficult to respond to the environment under real-time constraints. Since the Xilinx Virtex II Pro FPGA has the partial reconfiguration features, it can provide parallel and pipeline processing for image processing and matching. A novel reconfiguration architecture for this purpose is also described in this paper.

This paper is structured as follows. Section 2 describes the related works to the reconfigurable robot system done by other researchers. Section 3 presents the agent-based embedded system architecture for robot navigation system, which includes hardware/software partition, intelligent agent model definition, and multi-agent system paradigm. Section 4 presents several dynamic reconfiguration modules. Section 5 describes some experimental results implemented on Virtex II Pro FPGA. Section 6 gives the conclusions and future works.

## II. RELATED WORK

There are some researches on reconfiguration for robot software. Gafni *et al* [1] proposed an architectural style for real-time systems, in which the dynamic reconfiguration is implemented by a synchronous control task in response to the sensed conditions. To handle run-time dependencies between components, [2] employ run-time analysis of interactions to selectively determine which interactions should be allowed to proceed, in order to reach a safe state for change, and which to block until reconfiguration completed. Cobleigh *et al* [3] presented a hierarchically self-adaptation models for the robot system to provide fault-tolerance. MacDonald *et al* [4] proposes some design options for dynamic reconfiguration with their service-oriented software framework. Stewart *et al* [5] proposes a programming paradigm that using domain-specific elemental units to provide specific guidelines to control engineers for creating and integrating software components by using port-based object.

There is another category for robot self-configuration, which means the physical characteristics of the robot is reconfigured on the fly in response to the needs of the task or environment. [6] [7] are examples, which present a self-configurable robot design with a distributed software architecture that follows the reconfiguration of hardware.

There are very few researchers working on dynamic reconfiguration for robot system implemented on hardware. However, hardware reconfigurable computing has grown to become an important and large field of research in computation community using FPGA. The approach of applying reconfigurable logic for data processing has been demonstrated in some areas such as video transmission,

image-recognition and various pattern-matching operations [8]. The fastest solution for reconfiguration is through context switching which is able to store a set of different configuration bitstreams and makes the context switching in a single clock cycle. [9] [10] propose contest switching FPGA architectures which emphasized on both proving fast context switching as well as fast random access to the configuration memory. This is important because you may want to change one or more of your configuration streams inside the FPGA at runtime.

The major innovative architecture design proposed in this paper is applying the multi agent paradigm to the reconfigurable embedded system on both software and hardware design, and establishing a framework by which hardware agents can be dynamically reconfigured to meet the system's real-time constraints. The proposed reconfigurable architectures can provide the consistency preservation as well as low reconfiguration latency and fast random access to the configuration memory.

## III. AGENT-BASED EMBEDDED SYSTEM ARCHITECTURE

### A. Intelligent Agent Model

An *Agent* is an independent processing entity that interacts with the external environment and the other agents to pursue its particular set of goals. The agent pursues its given goals adopting the appropriate plans, or intention, according to its current beliefs about the state of the world, so as to perform the role it has been given. Such an intelligent agent is generally referred to as a Belief-Desire-Intention (BDI) agent. Under the BDI model, agents may be given "pre-compiled" behaviors, or they may plan or learn new plans at execution time. Giving BDI agent pre-compiled plans is a method for ensuring predictable behavior under critical operational conditions, and for ensuring performance.

BDI agents are highly suitable for the development of time and mission critical systems, as the BDI approach provides for the verification and validation of the model. The ability of intelligent agents to perform simple tasks autonomously has aroused much interest in the potential robotic application. Key characteristics of intelligent agents that make them attractive are: autonomy, high-level representation of behavior – easy to define command and control architectures, flexible behavior, combination of pro-activity and reactivity, real-time performance, and suitability for distributed applications

The agent control loop is: first determine current beliefs, goals and intentions, then find applicable plans, then decide which plan to apply, and finally start executing the plan. Both the software agents and hardware agents in our system adopted the BDI architecture model. The BDI agent can be expressed in a structure as the followings. This structure can be implemented by Verilog hardware description language (VHDL) on FPGA.

<Agent>{<Beliefs>  
 Constraints; Data Structures;  
 <Desires>  
 Values; Condition; Functions;  
 <Intentions and Commitment>  
 Methods; Procedures;}

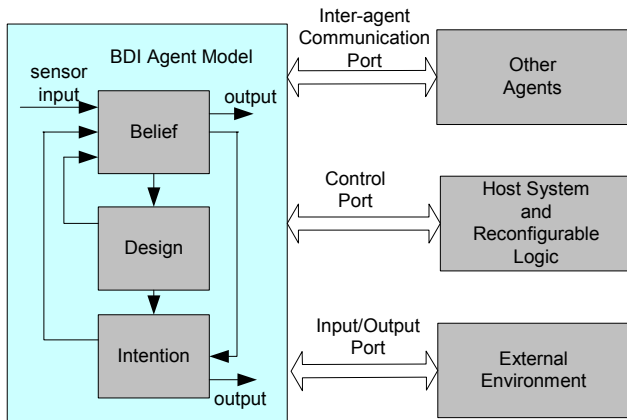


Fig. 1: The architecture of BDI agent model

In the model, beliefs and desires influence each other reciprocally. Furthermore, beliefs and desires both influence intentions. The architecture of the BDI agent model is shown in Fig.1, which has three external ports: inter-agent communication, control, and input/output. The control port is used for the system to activate/deactivate the agent, and for the agent to inform the system when it finishes its job, and also a clock signal for the agent to synchronize with the system. The input/output port is used to send and receive information to and from the host environment. The inter-agent communication port allows the agents to send/receive information with other agents and cooperate with each other.

### B. Multi-agent System Architecture

Hardware/software co-design provides a way of customizing the hardware and software architectures to complement one another in ways which improve system functionality, performance, reliability, survivability, and cost/effectiveness. In our agent-based embedded system, in general, more regularly structured agents that have highly repetitive and extensive time consuming operations are suitable for implementation in reconfigurable hardware, whereas the more complex and irregularly structured agents should be programmed in software.

Xilinx Vertex II Pro™ FPGA is adopted as the prototyping platform in our system. The PowerPC embedded in the FPGA is operating as a general purpose processor (RPP), and the reconfigurable fabric is used as a reconfigurable co-processor (RCP). The agents implemented in RPP are software agents, while the agents implemented in RCP are hardware agents. The agents implemented in hardware are placed in the configuration

memory of the reconfigurable system. Multiple agents can reside in a single configuration memory or they can reside in different configuration memories of the system simultaneously.

The multi-agent architecture for robot navigation system is shown in Fig. 2. Each sensor is represented by an individual BDI agent. Each agent is independent to each other and can communicate to each other. The sensor manager agent would analyze the sensor information sent by all the sensor agents, fuse the sensor data if necessary, and send the fused sensor data to path planning agent, obstacle avoidance agent, and emergency stop agent. Then those agents make decisions according to their algorithms and send the command to the actuator manager agent to take actions through the motion agents.

Based on the above multi-agent architecture, when a mobile robot navigates to different environments or sensors encounter some malfunctions, only the sensor agents and sensor manager agent need to be reconfigured. The other agents will stay in the same configuration status as long as they don't change their corresponding algorithms, which is usually the case for the high-level planning. In some cases such as moving from a flat floor to a rough terrain, the navigation becomes 3D instead of 2D, the path planner agent will have to deploy a new 3D algorithm. Since task planner agent is implemented in software agent, it will need software reconfiguration. Since software reconfiguration is not the focus of this paper, and there are some researches have been done to reconfigure robot software dynamically [4]. We will only focus on the hardware reconfiguration in this paper.

## IV. HARDWARE AGENTS RECONFIGURATION MODULES

Performing reconfiguration on a running system is an intrusive process. Reconfiguration may imply, for example, interference with ongoing interactions between entities. One of the main issues of dynamic reconfiguration is consistency preservation [11]. Changing management functionality must assure that system parts that interact with entities under reconfiguration do not fail because of reconfiguration, i.e., system consistency needs to be preserved. The consistency preservation could become a very complicated problem. In order to preserve the consistency while keep the dynamic reconfiguration procedure simple, we make the following assumptions for the reconfiguration models.

- 1) Agents work independently.
- 2) Agents can communicate with other agents.
- 3) Each agent can have a finite number of schemes which is a description of possible events. Each scheme of the agent can be created a priori before the application is implemented, and can be stored in the external memory.

Based on the above assumptions, several agent reconfiguration modules are proposed in this section.

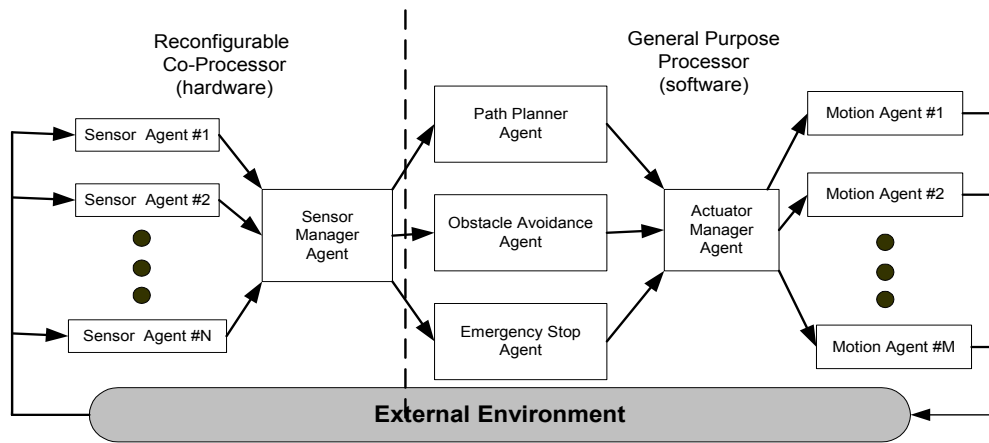


Fig. 2: Multi-agent architecture for robot navigation system

## A. Reconfiguration Models

### A.1. Virtual Agent Reconfiguration

This method is based on the work of Sckanina et. al [12] by implementing a user defined FPGA inside an ordinary FPGA. In this paper, as shown in Fig. 3, it is extended to include a number of *virtual* states that each agent could possibly go through at runtime. These virtual states are defined as configuration bitstreams which are loaded into the reconfigurable hardware memory when the system starts up. Thus, in this scheme the FPGA configuration registers are fixed at runtime. The device is reconfigured only by simply selecting an active state by a multiplexer based on its local point of view. There is no reconfiguration latency due to static reconfiguration. If the device contains plenty number of agents, the cost of this method is extremely high and the efficiency is very poor because all possible agent states must be implemented which requires more reconfiguration resources. However, the number of logic blocks in the Vertex blocks is large (for example, XC2VP30 has 30k logic cells) and each block can be configured in a set of ways. Therefore, if the number of sensor agents in the robot navigation system is very limited, and the computation of sensor data processing is not extremely intensive, this reconfiguration method is suitable for robot navigation system due to its low overhead and low reconfiguration latency.

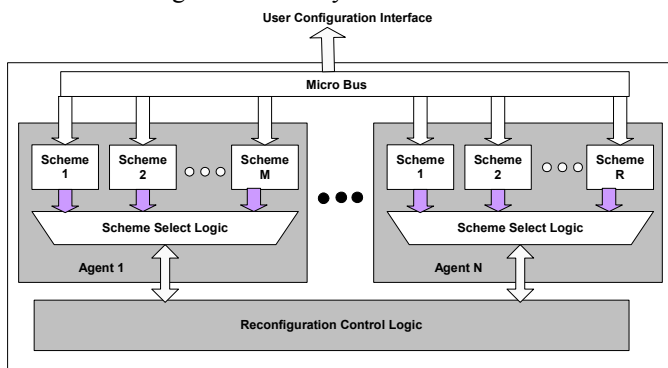


Fig. 3: Implementing a virtual agent reconfiguration

### A.2. Pipelined Redundant Agent Reconfiguration

In order to be able to use the reconfigurable resource more efficiently for the case that the number of reconfigurable agents is large, one option is that the replacement agent reconfiguration overwrites the previous agent state while suspending the functionality of the reconfigurable logic. This mode supports several temporal modes of operation. However, there is high reconfiguration latency because the system must spend lots of time to load the selected agent states from high capacity/low cost hardware configuration storage to the partially reconfigurable logic. This method is only suitable for those applications that do not have critical real-time constraints, and is not appropriate for robot navigation system.

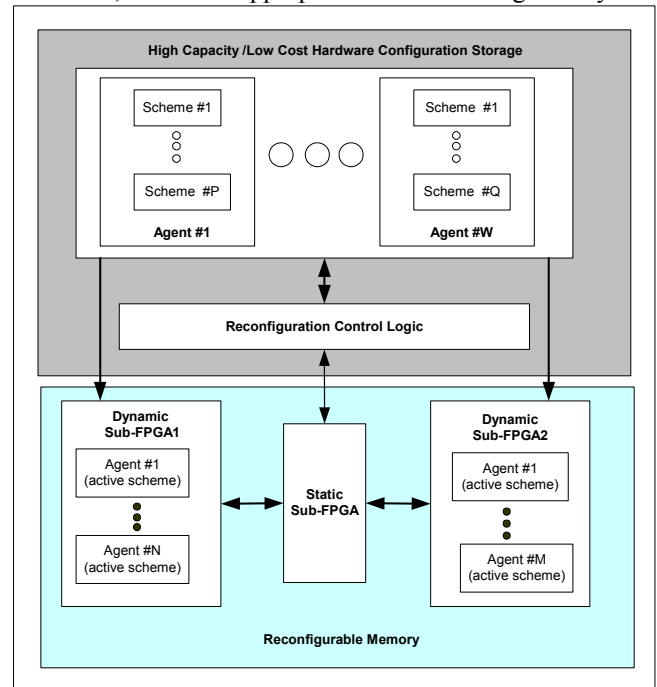


Fig. 4: Pipelined redundant reconfiguration architecture

Instead of waiting for the long latency time loading configurations from low-cost hardware to the reconfiguration memory, the FPGA can be partitioned into a set of sub-FPGAs,

and each is configured separately using partial reconfiguration. The static sub-FPGA is fixed and always active at runtime. This unit is managing the dynamic sub-FPGAs which are configured at runtime and are not always in operation. That is, only one of the dynamic sub-FPGAs is active at any given time. Thus, the one not being active can load its new configuration through reconfiguration control logic by partial reconfigurations, while the other one is executing. In this way, the FPGA becomes a pipelined unit where the configuration is pipelined with execution. The number of dynamic sub-FPGAs could be increased if the execution time is shorter than the configuration time. By applying this pipelined method, the reconfiguration latency is greatly diminished, which is very important for the real-time constraints for robot navigation system. However, only part of the reconfigurable resources can be used at a given time. This method can obtain the optimized balance between the reconfiguration latency and resource occupancy.

### B. Reconfiguration Architecture for Image Processing

For robot navigation system, a large amount of images from vision system need to be processed under real-time constraints. Image matching is critical for navigation system since 3D environment knowledge can only be obtained through multiple views. With the partial reconfiguration feature of Xilinx Virtex-II Pro FPGA, it is possible to make the image loading and image matching to execute at the same time. This parallel scheme can reduce the implementation time dramatically since the image loading from external memory to reconfigurable hardware is usually extensively time consuming. Fig. 5 shows this architecture where two buffers created, one is for loading and one for matching. When the image is loaded onto the buffer from I/O, the other buffer is applied for high-speed matching algorithm. When the image loading is finished, the control logic will check the status of another buffer. If the matching is done, the two buffers exchange operations. Otherwise, the control logic will inform the loading buffer to wait till the other buffer finishes the matching, and then exchange the operations.

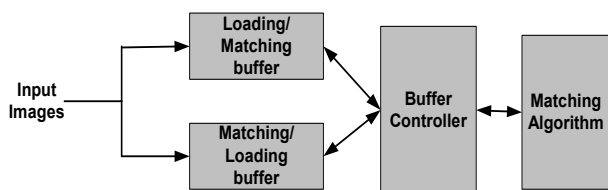


Fig. 5: A reconfigurable architecture for image matching

## V. EXPERIMENTAL RESULTS

In order to evaluate the proposed architecture design, a hybrid software and reconfigurable hardware co-design is necessary. It calls for a unified agent-based and hierarchical design paradigm encompassing:

1. *Software*: agent-based design in C++ implementing on general processors
2. *Hardware*: agent-based hardware description language (HDL) code, e.g. Verilog, on FPGA
3. *Interface*: agent-based software interface drivers and/or HDL decode/encode interface modules

The Xilinx ML310 embedded development platform is used as our prototype platform. The ML310 offers designers a Virtex-II Pro XC2VP30-based embedded platform. It provides 30k logic cells, over 2.4kb BRAM, and dual PPC405 processors, as well as onboard Ethernet MAC/PHY, DDR memory, multiple PCI slots, and standard PC I/O ports. Virtex II devices feature a regular architecture that comprises an array of CLBs surrounded by programmable IOBs, all interconnected by a rich hierarchy of fast, versatile routing resources. Software agents can be implemented on dual PPC405 processors while hardware agents on FPGA.

One advanced feature provided by Virtex-II Pro FPGA is active partial reconfiguration. Instead of resetting the device and performing a complete reconfiguration, new data is loaded to reconfigure a specific area of a device, while the rest of the device is still in operation. For Virtex-II Pro FPGA, data is loaded on a column-basis, with the smallest load unit being a configuration bitstream “frame”. Module-based partial reconfiguration is accomplished in slave SelectMAP mode in the robot navigation system application.

In our mobile robot navigation system, one CCD camera, one laser rangefinder, one night vision gear (NVG), one GPS receiver, and encoder odometry are installed to a Pioneer 3DX robot as on-board sensors for navigation system. The ML310 embedded platform is attached to the robot as an on-board processor. Each sensor is designed as one hardware agent and sensor management as another hardware agent to fuse the sensor data, all of them are implemented on reconfigurable FPGA. Path planning, obstacle avoidance and emergency stop are implemented on PPC405 processors as software agents.

Here is the experimental scenario: when the robot moves around in an indoor office with good illumination, only CCD camera and laser are activated for obstacle detection, odometry is activated for robot positioning. When it moves to a dark outdoor parking lot at night, the CCD camera is deactivated while the NVG is activated, and GPS is activated with the odometry.

By implementing the first proposed virtual reconfiguration architecture, all the possible states of each sensor have to be listed in the FPGA reconfigurable memory, which means only one reconfigurable module is needed. To simplify the problem, some threshold values are set for each sensor to make them active or passive. When the real sensor value is higher than the threshold, the corresponding sensor is activated, otherwise, it is deactivated. The sensor management agent then fuses all the activated sensor data to inform all the higher-level software agents.

For the pipelined reconfiguration architecture, four reconfiguration modules are implemented, indoor with lights on, indoor with lights off, outdoor during daytime, and outdoor at night. Each module includes all the possible active

sensor agents. If working environment is changed, the corresponding module is loaded to the reconfiguration memory. The sensor management agent is designed as static sub-FPGA which only takes one active sub-FPGA at a time. In order to preserve the consistency, the sensor management agent does not reply on the state of the module under reconfiguration while reconfiguration is taking place. In order to ensure proper operation of the design during the reconfiguration process, explicit handshaking (e.g., module ready/not-ready) logic is implemented. Since the state of the storage elements inside the reconfigurable module are preserved during and after the reconfiguration process, this “prior state” is utilized as the current state to the sensor management agent when the reconfiguration is taking place. We can also take advantage of this “prior state” information after a new reconfiguration is load as a potential learning capability.

The original idea of the evaluation of our proposed architecture was to compare the computation time implementing the reconfiguration in software only with general processors and in software/hardware co-design with Virtex-II FPGA. However, implementing the reconfigurable software agents could be another totally different research project with lots of challenges [3][4]. Therefore, Both proposed reconfiguration architectures are implemented successfully on the Pioneer 3DX robot navigation system with only the software/hardware co-design method. Table I shows the resource usages of these two models. It is apparently that the pipeline mode uses less CLB slides but more I/Os and Function generators than the virtual one.

TABLE I  
RESOURCE USAGES OF HARDWARE AGENTS

Resources	Virtual Model	Pipelined Model
	utilization	utilization
I/Os	14.20%	18.34%
Function Generators	23.12%	32.87%
CLB slides	36.94%	18.23%

We also verify the proposed architecture for image processing using FPGA. The image data captured by the camera is sent to FPGA through PCI bus. Under the first environment, the image processing and matching with and without the proposed reconfigurable architecture for image matching are implemented. The one with the proposed architecture is about 10 times faster that the other one.

## VI. CONCLUSION AND FUTURE WORKS

This paper proposes a dynamic self-adaptive mobile robot system by applying an agent-based reconfigurable real-time embedded platform. The multi-agent system paradigm has been applied to implement the reconfiguration system, which provides the same methodology for software and hardware development and simply the communication and

reconfiguration procedure without building a new structure for the interface between the software and hardware. Another advantage of our platform is that the real-time constrains problem in the robot system can be efficiently resolved by implementing the sensors processing and sensor fusion on a reconfigurable hardware FPGA, which is much faster than implemented by software only. This proposed embedded platform can be easily applied to other robotic applications, such as server robots, space robots, and even multi-robot systems.

Although the focus of this paper is on the hardware reconfiguration of the robot system, the software reconfiguration is also critical if the software agent has to change its algorithm during runtime. When both the software agents and hardware agents have to be reconfigured simultaneously, it will be difficult to preserve the consistency and synchronization for the reconfiguration. However, our multi-agent architecture provides a feasible way for this problem because due to the shared methodology. We will investigate these issues in our future work.

## REFERENCES

- [1] V. Gafni, “Robots: a real-time systems architectural style”, In *Proc 7<sup>th</sup> European software engineering conference*, pages 57-74, Toulouse, 1999.
- [2] J. Almeida, M. Wegdam, M. Sinderen, and L. Nieuwenhuis, “Transparent Dynamic Reconfigurable for CORBA”, *Proceeding of the 3rd International Symposium on Distributed Objects & Applications (DOA 2001)*, Sept. 17-20, 2001, Rome, Italy.
- [3] J. Cobleigh et al, “Containment units: a hierarchically composable architecture for adaptive systems”, *ACM SIGSOFT Software Engineering Notes*, 27(6):159-165, November 2002.
- [4] B. MacDonald, B. Hsieh, and I. Warren, “Design for Dynamic Reconfiguration for Robot Software”, *2<sup>nd</sup> International Conference on Autonomous Robots and Agents*, December 13-15, 2004, Palmerston North, New Zealand.
- [5] D. Stewart, R. Volpe, and P. Khosla, “Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects”, *IEEE Trans. On Software Engineering*, vol. 23, no. 12, December 1997.
- [6] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, “Self-reconfigurable modular robots – hardware and software development in aist”, In *Proceedings, IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, Volume 1, pages 339-346, October 8-13, 2003.
- [7] S. Patterson, K. Knowles, and B. E. Bishop, “Toward Magnetically-Coupled Reconfigurable Modular Robots”, *Processing of IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004.
- [8] J. Villascnor and W.H. Mangionc-Smith, “Configurable computing”, *Scientific American*, no. 6, 1997.
- [9] S. M. Scalera and J.R. Vazques, “The design and implementation of a context switching FPGA”, *IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 78-85.
- [10] L. Scanina and R. Ruzicka, ‘Design of the special fast reconfigurable chip using common FPGA’, in *Proc. Of Design and Diagnostics of Electronic Circuits and Systems – IEEE DDECS’2000*, 2000, pp.161-168.
- [11] J. Kramer and J. Magee, “The Evolving philosophers problem: dynamic change management”, *IEEE Trans. Software Engineering*, 16(11):1293-1306, 1990.
- [12] L. Scanina and R. Ruzicka, “Design of the special fast reconfigurable chip using common FPGA”, in *Proc. Of Design and Diagnostics of Electronic Circuits and Systems – IEEE DDECS’2000*, 2000, pp 161-168.