ORIGINAL ARTICLE SPECIAL ISSUE: Intelligent Robotics(2/2) Guest edited by Yi-Hung Liu

# ESAIR: A Behavior-Based Robotic Software Architecture on Multi-Core Processor Platforms

**Chin-Yuan Tseng*, Yu-Lun Huang, and Jwu-Sheng Hu**
*Institute of Electrical and Control Engineering, National Chiao-Tung University, Taiwan*
(**Received** 7 October 2012; **Accepted** 22 October 2012; **Published on line** 1 March 2013)
*Corresponding author: esid.ece96g@nctu.edu.tw
DOI: 10.5875/ausmt.v3i1.167

**Abstract:** This paper introduces an Embedded Software Architecture for Intelligent Robot systems (ESAIR) that addresses issues with parallel thread executions on multi-core processor platforms. ESAIR provides a thread scheduling interface to improve the execution performance of a robot system by assigning a dedicated core to a running thread on-the-fly and dynamically rescheduling the priority of the thread. We describe the object-oriented design and the control functions of the ESAIR. The modular design of ESAIR helps improve software quality, reliability, and scalability in both research and practical settings. We demonstrate this improvement by realizing ESAIR on an autonomous robot named Avatar. Avatar implements various human-robot interactions, including speech recognition, human-following, face recognition, speaker identification, among others. With the support of ESAIR, Avatar can integrate a comprehensive set of behaviors and peripherals with better resource utilization.

**Keywords:** Behavior-based software design; Human-robot interaction; Multi-core platform; Software architecture

## Introduction

A significant challenge in designing robot systems is the software architecture, especially for those applications requiring real-time or complex human-computer interactions. A robot can be viewed as an automatic machine that independently integrates different hardware and software components developed separately. Due to a lack of specifications and tightly-coupled designs, robot developers usually need to redesign the hardware platform and the software architecture for every new system; this is especially true for cases that use embedded platforms on which both software and hardware resources are limited. In addition, the rapid growth of robotic systems is leading developers to introduce a variety of applications to tackle diverse requirements. To reduce the length of the development process, software engineering is widely adopted in designing robot systems. Recently, many researchers have proposed various software architectures in the development of their robotic systems, such as CARMEN [1], Player [2, 3], CLARAty [4], MARIE [5, 6], MIRO [7], OROCOS [8, 9], ORCA [10, 11], and RT-Middleware [12]. Most of them provide a framework for robot systems employing reusability, flexibility, modularity and distributed processing, when analyzed as component-based software architecture.

While those studies have provided several frameworks to solve the problems of developing software modules for robots, the majority do not address issues resulting from resource management in robots that adopt multi-core processors. Although multi-core processors can provide real parallelism by executing multiple tasks on different cores simultaneously, managing resources among the parallelized tasks is an important consideration to realize the design of such a robot on a multi-core platform. A parallel design can leverage the advantages of a multi-core platform such that calculations may be carried out simultaneously to achieve better performance.

In 2007, we published the design basis of our Embedded Software Architecture for Intelligent Robot (ESAIR) systems [13, 14]. The design philosophy considers the functional components within a robot system as well as the capabilities and behaviors that can be achieved by the robot. Generally, ESAIR aims to offer a heterogeneous computing environment, and provides three major functionalities: 1) a pluggable programming interface for agent programs such that they can register or hook themselves up to the control center; 2) a discovery mechanism for agent programs to find and drive necessary software objects within the same robot system; and 3) a communication interface to talk to other agent programs in different robot systems.

In this paper, we enhance the design to further provide resource management for robots using multi-core processors. With our newly proposed framework, it is easier for programmers to customize specific software modules for an individual robot system and configure these modules according to the limitations of the computing resources of that robot. The modular design of ESAIR provides reusability, flexibility and distributed processing to robot development. The parallel design and priority assignments in particular make ESAIR adaptable to advanced multi-core computing platforms.

The remainder of this paper is organized as follows. Section II discusses related work in the area of developing robot software frameworks. Section III details the design and implementation of the proposed software architecture for intelligent robots, ESAIR, from a class-based point of view. Section IV and V present the design features of our navigation robot supporting Simultaneous Localization and Mapping (SLAM) and the implementation of ESAIR on Avatar, respectively. We conclude the paper in Section VI.

## Related Work

The development of various robot software frameworks has different objectives such as simplification of the development process, reusability, integration, and flexibility. In addition, inter- and intra-robot communication is another important consideration for robot software frameworks. Some frameworks rely on middleware to handle the communication between software components, such as MIRO, Player/Player2, ORCA, MARIE, OROCOS, etc. Other projects, including CLARAty, CARMEN, and others use socket-based communication methods for message passing. Most of these methods utilize GIOP, IIOP, or RPC as their underlying transport protocol. With these communication methods, researchers are able to run robot software frameworks on distributed systems or multi-core processors for distributed processing [15].

In 2007, the Stanford Artificial Intelligence Laboratory developed a Robot Operating System (ROS) with a comprehensive set of libraries and tools to help developers create robot applications. ROS is an open-source, meta-operating system on top of a heterogeneous computer cluster, and it provides some general OS primitives, including for hardware abstraction, low level device control, message-passing between processes, and package management. The libraries of ROS also support Unix-like OSs such as Ubuntu, Fedora, Gentoo, Arch Linux, and other Linux platforms [16], allowing for cross-platform development.

DROS (Dave's Robotic Operating System), developed by David Austin in 2002, is another framework for robot systems [17]. It is open-source and distributed under the GNU Public License. Since different robot systems often have different requirements, such as robustness and real-time performance, the MSDE (Modular Software Development Environment) of DROS aims to provide a robust framework for the development of modular software. The communication library in DROS, GeneralComms, is designed such that the framework is compatible with Unix-like operating systems.

Since multiple microcontrollers are commonly adopted in advanced robot systems, Magnenat *et al.* [18, 19] proposed ASEBA, an event-based middleware that supports distributed control and resource exploitation in a multi-microcontroller robot system. In ASEBA, inter microcontroller communications can be sent via asynchronous events without the participation of the main processor.

In addition to the embedded microcontroller systems, some robot systems adopt a multi-core processor to support heavy-load computing tasks. In 2004, MFSM (Modular Flow Scheduling Middleware), an open source architecture middleware supporting a multi-threaded environment for the SAI (Software Architecture for Immersipresence) project, was proposed to provide robust real-time vision for robots [20]. MFSM offers a generic extensible data model to allow encapsulation of existing data formats and standards as

**Chin-Yuan Tseng** is with the Department of Electrical and Computer Engineering, National Chiao-Tung University, Hsinchu, Taiwan.

**Yu-Lun Huang** is with the Department of Electrical and Computer Engineering, National Chiao-Tung University, Hsinchu, Taiwan.

**Jwu-Sheng Hu** is with the Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, and Mechanical and Systems Research Laboratories, Industrial Technology Research Institute.

well as low-level service protocols and APIs. Such a model simplifies the inter-operation between processes. With the parallel support of MFSM, SAI can directly take advantage of a robot using a multiple-core platform for its asynchronous concurrent processing while synchronizing data streams.

In 2009, Niemueller *et al.* [21, 22] proposed Fawkes, a component-based robot software framework. Fawkes was designed to address the considerations of realizing a robot design on a multi-core computing platform. With its proposed communication middleware and programming interfaces, Fawkes offers a hybrid blackboard/messaging infrastructure and run-time loadable plug-in mechanisms based on the concept of component-based software engineering.

However, none of the above robot operating systems or software frameworks considered the requirements of resource management, extensibility, multi-threading, scheduling, etc., that are essential in the development of a robot system.

## The Proposed Framework: ESAIR

The design philosophies behind ESAIR are based on behavior-based robotics, which consists of three major components: sensors, behaviors, and actuators. ESAIR extends the concept of behavior-based robotics, and further decomposes a robot system into four basic components: Resource, Connection, Behavior, and Supervisor:

1) '**Resource**' manages the I/O peripherals of a robot system, such as sensors and actuators.
2) '**Behavior**' computes data gathered from sensors and performs responses on actuators. It works as the "brain" of a robot system to determine robot behavior.
3) '**Connection**' behaves as a mediator between the Behavior and Resource components, and centrally manages data exchanges.
4) '**Supervisor**' manages the load/unload of components and coordinates their execution.

ESAIR implements the above components in a base class and defines its member functions. Nine derived classes are extended from the base classes to define the data/configuration interface between components, as shown in Figure 1. Only classes derived from the Resource and Behavior classes can be extended by a user to redefine the operations of a component. ESAIR treats each class as an individual thread, which can be executed and scheduled separately.

Figure 2 shows the software architecture of Avatar, where ESAIR is used to coordinate the HRI software modules, including SLAM, navigation, etc.

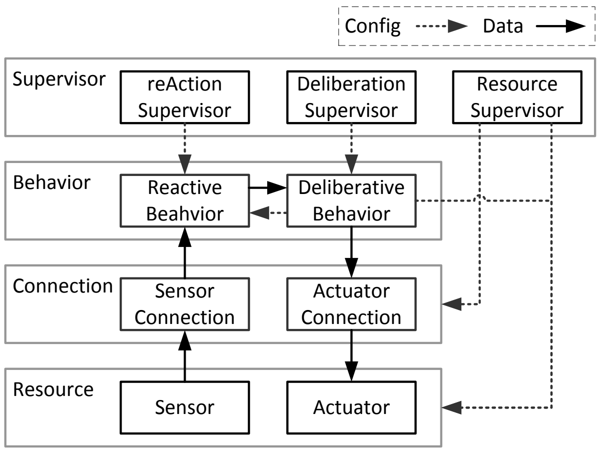


Figure 1. The class design of ESAIR.

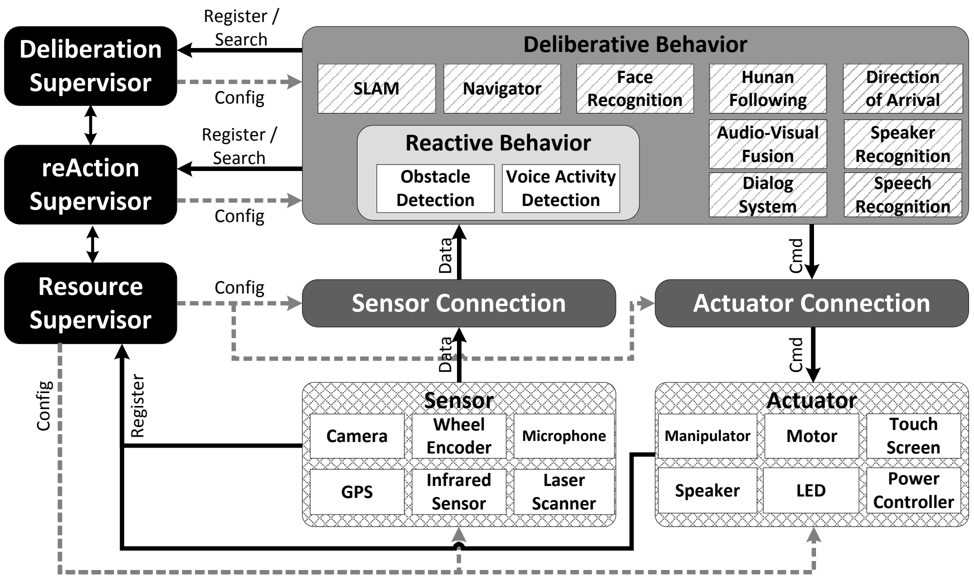


Figure 2. (a) Avatar robot design, (b) Robot application in ESAIR.

## *Class Design of ESAIR*

In this section we describe the design of the classes used in ESAIR.

1) **Resource Class:** The Resource class consists of a variety of hardware resources used in robotic systems. It interacts with drivers and manipulates data before transferring them to a Connection class. The Sensor (S) and Actuator (A) classes derived from the Resource class are distinguished by the direction of the data flow. 'S' acts as an agent that interacts with a sensor and manages the sensed data. The data are sent to a Sensor Connection class and dispatched to a Behavior class. 'A' performs the behaviors generated from Behavior classes. It receives a command through an Actuator Connection class and interacts with the hardware.

2) **Connection Class**: The Connection class bridges the Behavior and Resource classes. It conceals the implementation details of heterogeneous hardware devices and provides the Behavior classes with a unique communication interface. To cooperate with the classifications of the Resource classes, a Connection class is also divided into two derived classes: the Sensor Connection class and the Actuator Connection class. The Sensor Connection class is a buffer between the Sensor classes and Behavior classes to help minimize the cost on data synchronization. Similarly, the Actuator Connection class works as a multiplexer to pick up commands from a Behavior class and transfer them to an Actuator class.

3) **Behavior Class**: The Behavior class determines an action taken by a robot. Based on the classification of behavior-based robotics [23], the Behavior class is extended to the Reactive Behavior class and the Deliberative Behavior class.

- The **Deliberative Behavior class** defines a sequence of calculated behaviors, and performs optimized operations via monitoring the input data periodically. A Deliberative Behavior class can also be used to configure the settings of other Resource and Behavior classes at run time. Such a design helps coordinate the behavior of a robot.
- The **Reactive Behavior class** is responsible for performing immediate responses to the environment. A memory-less algorithm or a simple I/O relationship table is implemented in the class to quickly generate a behavior command. Since the design of the Reactive Behavior class focuses on those behaviors with critical time-constraints, the command generated by a Reactive Behavior class usually has a higher priority.

4) **Supervisor Class**: The Supervisor class manages plug-in interfaces for searching, inserting, and deleting components. Three classes inherit from the Supervisor class: the Reaction Supervisor class, the Deliberation Supervisor class, and the Resource Supervisor class. A Resource Supervisor class controls the Resource and Connection classes, while the other two Supervisor classes manage the Deliberative Behavior and Reactive Behavior classes, respectively. A Deliberation Supervisor class provides an extra interface to forward configuration commands from Deliberative Behavior classes to Resource classes. Thus, a Deliberative Behavior class can control the execution or change the settings of the Reactive Behavior, Sensor, and Actuator classes.

## *Resource Management in ESAIR*

The "brain" (CPU) of a robot system is in charge of manipulating computationally intensive algorithms and coordinating peripherals. In recent years, the trend in the development of CPUs has been to integrate multiple independent cores into a single core. This is termed a multi-core processor, and includes the common ARM MPCore [24] and Intel Core processor family [25]. This new architecture fulfills the requirement for robot developers needing lower power consumption and higher computing ability. Traditionally, task scheduling is applied on a single-core processor for pseudo–parallelism, which simply runs each process in turn for a short time period. The advance of new multi-core technology extends the computing system to real parallelism by running multiple processes on different cores simultaneously. ESAIR focuses on robot systems using multi-core processors to provide an interface for better resource utilization on a multi-core platform. We propose a thread scheduling interface to improve the performance of the executing threads by assigning each thread to a specified core and setting its priority. The following paragraphs detail the core assignment and priority assignment processes defined in ESAIR.

### Core Assignment

The performance of a robot system highly depends on the workload of a CPU, especially for computationally intensive components. In ESAIR, we define a thread-scheduling API that specifies a dedicated core to serve a computationally intensive thread. With the proposed API, we bind critical threads to specified cores, so they can be executed simultaneously on the multi-core platform and take advantage of the true parallelism of the hardware. In addition, 'core assignment' can be dynamically adjusted according to changes in running tasks. Such a design helps a robot system obtain better stability when running critical threads.

### Priority Assignment

We also consider time-critical threads which should respond in near real-time during urgent events. Although ESAIR does not support real-time responses, we still design an interface for assigning priority of a thread, such that urgent events can be executed faster by dynamically boosting the execution priorities. The thread-scheduling API guarantees that a priority-boosted thread can complete its job earlier than if running at the normal execution level. In ESAIR, threads can also be set to run with priorities below the normal execution level as necessary. For example, a robot performing human-following can lower the priority of a voice command process, since a delayed command interpretation is more acceptable compared with losing the tracking target.

Both Supervisor and Deliberative Behavior classes implement a control interface to adjust robot performance on-the-fly through a dedicated configuration algorithm. With the design of thread-scheduling interfaces, ESAIR realizes a manual management of resources on a robot platform.

## Avatar: Design of SLAM and Navigation

We have designed Avatar, an autonomous robot that supports a comprehensive set of human-robot inter-actions (HRI) including SLAM, navigation, speaker identification, face recognition, speech recognition, human-following, among others. Avatar adopts a dual-core CPU as its main processor and hosts a variety of peripherals (see Figure 3), such as:

1) dual cameras for robot vision,
2) two microphone arrays and a pair of speakers for audio data, and
3) one laser scanner, two wheel encoders, and two wheel controllers for robot movements.

Figure 2 shows the software architecture of Avatar, where ESAIR is adopted for coordinating the HRI software modules, including SLAM, navigation, etc.

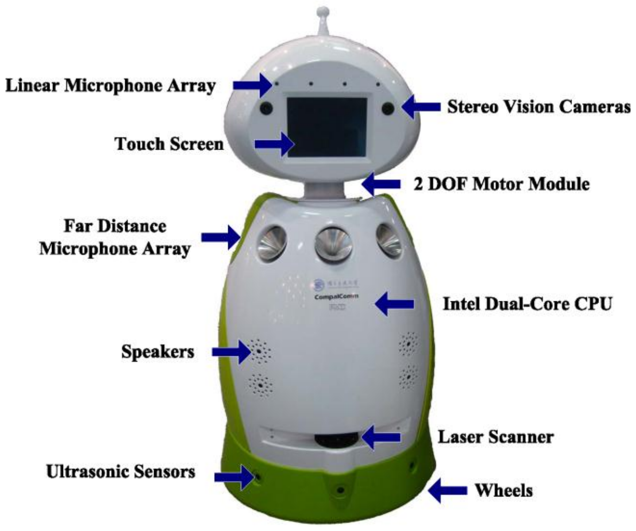


Figure 3. Avatar robot design.

### *Hardware Design*

Avatar is designed for applications of image processing, speech processing, sensor fusion, and HRI, among others. In Avatar (see Figure 3), a touch screen, laser scanner, odometer, and wheels are used to implement SLAM and the navigation mentioned above. For example, the laser scanner installed in Avatar uses Sick S200 with a 270° scanning angle and a 1.5 m range to scan and determine the distances to subjects within a room. The odometer indicates the distance traveled by Avatar and can be used to estimate the location of Avatar relative to a starting point. With the laser scanner and odometer, Avatar can construct a 2D grid map for subsequent navigation. We programmed the SLAM and navigation functions on an Intel dual-core processor.

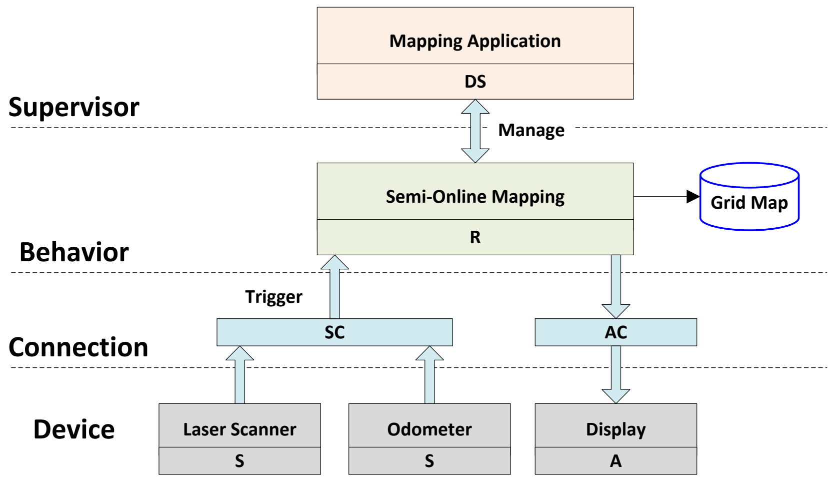


Figure 4. Software design architecture for constructing a 2D grid map in Avatar.

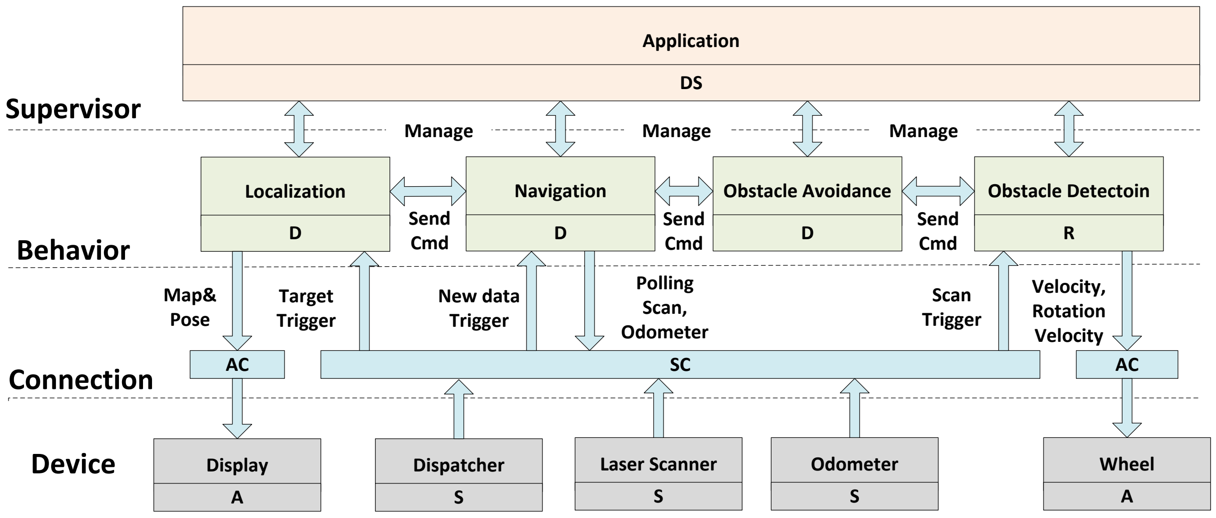


Figure 5. Software design architecture of SLAM and navigation in Avatar.

### *Software Design*

With ESAIR, we implement several software modules for Avatar, including localization, mapping, navigation, and obstacle detection. Figure 4 shows how the 2D grid map is constructed, while Figure 5 shows the design of the SLAM and navigation. Since ESAIR provides supervisor and connection libraries, we need only focus on the implementation of classes at the behavior and device layers.

Before Avatar can navigate in a room, a 2D map is required for its reference. We designed a semi-online mapping algorithm to build up such a 2D grid map at the behavior layer, as shown in Figure 5. The semi-online mapping algorithm detects the subject distances and performs scan-matching to construct a 2D grid map of the room. According to the definition of Reactive Behavior, the semi-online mapping algorithm checks whether the incoming action is an immediate action or not: thus, we implement such a function with the Reactive Behavior class of ESAIR. As mentioned previously, the laser range finder and odometer are used to provide the subject distances and traveling distances of Avatar, and therefore are implemented with the ESAIR Sensor class. With Avatar, a display interface is also implemented to show the 2D grid map and the current position of Avatar. The display function takes the subject distances as its input parameter, so we realize such a function with the ESAIR Actuator class.

The behavior layer shown in Figure 5 represents the localization function of Avatar. The localization function collects sensor data from the laser scanner for a period of time, so that it can perform scan matching and determine the current location of Avatar. Since it takes time for such a behavior to calculate and obtain the current location of the robot in the 2D grid map, we program the localization function with the ESAIR Deliberative Behavior class. Similarly, the navigation function updates the current location of the robot, follows the routing path, and receives the signal from the obstacle avoidance function. Hence, we realize the navigation function with the Deliberative Behavior class of ESAIR.

The obstacle avoidance function is required when Avatar moves within a room, so that it is able to avoid obstacles on its routing path. An obstacle detection function is executed to detect obstacles and alert Avatar to the danger of collision with them. The ESAIR Reaction Behavior class is then used to realize the obstacle detection function.

To supply the data required by the classes at upper layers, the Sensor classes at the device layer are used to handle the sensor data collected by the odometer and laser scanner and other hardware. In addition, we implement a dispatcher using the ESAIR Sensor class to recognize a user command and assign the command to a specified target. In Figure 5, two primitive Actuator classes are used to realize a display function and a velocity function:

– The **display function** shows the location of Avatar.
– The **velocity function** accepts a moving command and controls the wheel velocities accordingly.

## Implementation

The localization function ($D_1$) of SLAM periodically updates the location of Avatar in the map, after initially constructing the 2D grid map using laser sensors. The navigation function ($D_2$) provides a routing path for

Avatar to travel. When the obstacle detection function ($R_1$) finds that Avatar is close to an obstacle, it triggers the obstacle avoidance function ($D_3$) and navigation function ($D_2$) to prevent Avatar from hitting it. In this section, we briefly explain how the proposed ‘core assignment’ and ‘priority assignment’ help improve the performance of Avatar.

## *Core Assignment*

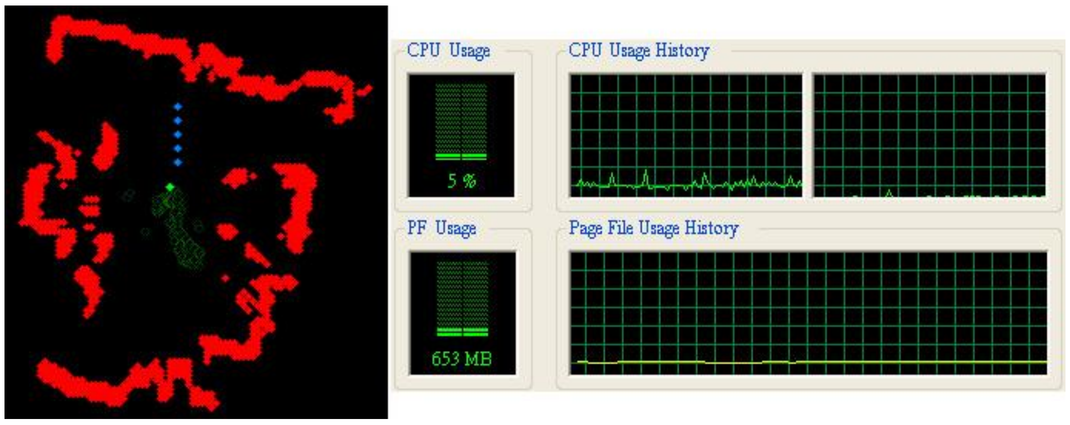
On a multi-core system, when multiple threads are scheduled by the default scheduler of the OS, cache misses may occur due to thread migration. Even if only one CPU core is used to dominate all working threads, cache misses still occur due to the size limitation of the cache. We conducted several experiments to show how ESAIR improves the performance of a robot system. In the first experiment, we used an Intel VTune Performance analyzer to record the L1 and L2 cache miss ratios when Avatar performed SLAM and navigation functions (Table 1). Here, we designed three cases to show variations in L1 and L2 cache misses. In the first case, threads were dominated by the default OS scheduler. We ran threads on a single core in the second case; while ESAIR core assignment was applied in the third case. The results clearly show that the average L1 cache miss ratio is the lowest when ESAIR core assignment is applied to Avatar. The L2 cache miss ratios are similar, since L2 caches are shared among CPU cores.

In the second experiment, we placed Avatar in a disorderly environment. The GUI of Figure 6 illustrates the boundary, robot location, and routing path in red, green, and blue, respectively. We also show the CPU and page file usages before and after applying ESAIR. In the experiment, we assigned sensor and actuator threads to Core1, and SLAM and navigation to Core2. Comparing the results in Figure 6 (a) and (b), it can be seen that the page file usage decreased after threads were specified to different cores using ESAIR.

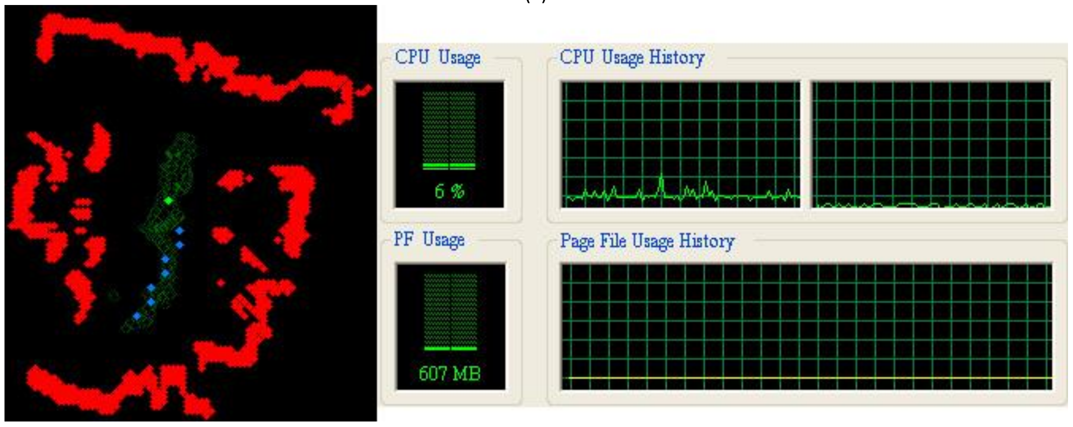
Table 1. Comparison of alternative scheduling systems.

| ***Analysis*** | ***Cache Miss Ratio*** | |
|---|---|---|
| | ***L1*** | ***L2*** |
| Case 1 | 72.074% | 67.856% |
| Case 2 | 71.944% | 67.054% |
| Case 3 | 68.074% | 65.520% |

| | |
|---|---|
| Case 1: | OS dominates the thread/task scheduling |
| Case 2: | All threads/tasks are run on a single CPU core |
| Case 3: | Core assignment is applied, assigning threads to dedicated cores |



(a)



(b)

Figure 6. CPU and page file usages: (a) before assigning threads to specified cores; and (b) after assigning threads to specified cores using ESAIR.

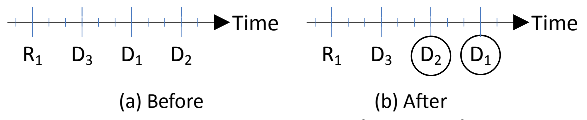


Figure 7. Operation system scheduling before and after reassigning thread priorities using ESAIR.

### *Priority Assignment*

The execution order of threads running in a robot system highly depends on the robotic application, and should be carefully prioritized. The ability to dynamically assign priorities for critical threads helps improve the performance of robotic applications. Figure 7(a) and (b) show examples of execution orders before and after the priority reassignment of $D_2$ and $D_1$ at run time, such that the navigation thread can be completed earlier by Avatar.

## Conclusion

In this paper, we proposed a behavior-based robotic software architecture (ESAIR) for a multi-core platform. ESAIR offers a framework that integrates many features in a robot system. We implemented an autonomous robot system named Avatar using ESAIR. In Avatar, SLAM and navigation functions were implemented to demonstrate improvements after ESAIR was applied. With ESAIR, a designer can efficiently program robot behaviors by applying core assignment and priority assignment. The results show that ESAIR can help integrate a comprehensive set of functions into a system, and improve software quality, reliability, and scalability in both research and practical settings.

## Acknowledgment

This work was supported in part by the National Science Council, Ministry of Economic Affairs, Ministry of Education, Taiwan, as well as other organizations.

## References

[1] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, Nevada, 2003, pp. 2436-2441. doi: 10.1109/IROS.2003.1249235

[2] T. H. J. Collett and B. A. Macdonald, "Player 2.0: Toward a practical robot programming framework," in *Australasian Conference on Robotics and Automation*, Sydney, Australia, 2005. Available: http://www.araa.asn.au/acra/acra2005/papers/collet.pdf

[3] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *The 11th International Conference on Advanced Robotics*, 2003, pp. 317-323. Available: http://robotics.usc.edu/~gerkey/research/final_papers/icar03-player.pdf

[4] I. A. D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin, "CLARAty and challenges of developing interoperable robotic software," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas. Nevada, 2003, pp. 2428-2435. doi: 10.1109/IROS.2003.1249234

[5] C. Cote, D. Letourneau, F. Michaud, J. M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sendal, Japan, 2004, pp. 1820-1825. doi: 10.1109/IROS.2004.1389661

[6] C. Côté, Y. Brosseau, D. Létourneau, C. Raïevsky, and F. Michaud, "Robotic software integration using marie," *International Journal of Advanced Robotic Systems,* vol. 3, no. 1, pp. 55-60, 2006. Available: http://cdn.intechweb.org/pdfs/4163.pdf

[7] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation,* vol. 18, no. 4, pp. 493-497, 2002. doi: 10.1109/TRA.2002.802930

[8] H. Bruyninckx, "Open robot control software: The Orocos project," in *IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2001, pp. 2523-2528 vol.3. doi: 10.1109/ROBOT.2001.933002

[9] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003, pp. 2766-2771. doi: 10.1109/ROBOT.2003.1242011

[10] A. Makarenko, A. Brooks, and T. Kaupp, "ORCA: Components for robotics," presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China, 2006. Available: http://www.cas.edu.au/content.php/237.html?publicationid=341

[11] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, "Towards component-based robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 163-168. doi: 10.1109/IROS.2005.1545523

[12] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and Y. Woo-Keun, "RT-middleware: Distributed component middleware for RT (robot technology)," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 3933-3938. doi: 10.1109/IROS.2005.1545521

[13] Y. l. Huang, C. y. Tseng, J. s. Hu, and K. h. Huang, "Design and implementation of a dual-robot system using ESAIR," *Journal of Software Engineering Studies,* vol. 4, no. 1, pp. 44-55, 2009.

[14] Y. L. Huang, E. C. Hsia, and J. S. Hu, "The design and implementation of an embedded software architecture for intelligent robots (ESAIR)," in *International Conference on Open Source*, Taipei, Taiwan, 2007.

[15] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics: A survey," in *IEEE Conference on Robotics, Automation and Mechatronics*, Chengdu, China, 2008, pp. 736-742. doi: 10.1109/RAMECH.2008.4681485

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009. Available: http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[17] D. Austin and L. Cole, *Dave's robotic operating system (DROS)*. [Online]. Available: http://dros.org/ [Accessed: September 30, 2012]

[18] S. Magnenat, V. Longcham, and F. Mondada, "ASEBA, an event-based middleware for distributed robot control," presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Diego, CA, 2007. Available: http://infoscience.epfl.ch/record/111860/files/aseba-iros2007-workshop.pdf

[19] S. Magnenat, P. Rétornaz, M. Bonani, V. Longchamp, and F. Mondada,"ASEBA: A modular architecture for event-based control of complex robots," *IEEE/ASME Transactions on Mechatronics,* vol. 16, no. 2, pp. 321-329, 2011. doi: 10.1109/TMECH.2010.2042722

[20] K. Kwangsu and G. Medioni, "Robust real-time vision for a personal service robot in a home visual sensor network," in *The 16th IEEE International Symposium on Robot and Human interactive Communication*, Jeju, Korea, 2007, pp. 522-527. doi: 10.1109/ROMAN.2007.4415142

[21] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework Fawkes," in *Simulation, modeling, and programming for autonomous robots*, vol. 6472, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. Stryk, Eds. Berlin/Heidelberg: Springer, 2010, pp. 300-311.

[22] T. Niemueller, "Developing a behavior engine for the Fawkes robot-control software and its adaptation to the humanoid platform Nao," M.S. Thesis, RWTH Aachen University, Knowledge-Based Systems Group, Aachen, Germany, 2009.

[23] R. C. Arkin, *Behavior-based robotics*. Cambridge, Mass.: MIT Press, 1998.

[24] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *Computer,* vol. 38, no. 7, pp. 42-50, 2005. doi: 10.1109/MC.2005.239

[25] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to intel® core™ duo processor architecture," *Intel® Technology Journal,* vol. 10, no. 2, 2006. doi: 10.1535/itj.1002.01