Applying Dynamic Software Architecture Management to Home Service Robot Software*

Dongsun Kim and Sooyong Park†
Department of Computer Science
Sogang University
Shinsoo-dong, Mapo-Gu, Seoul, Republic of Korea
{darkrsw, sypark}@sogang.ac.kr

Mun-Taek Choi and Munsang Kim
Center for Intelligent Robotics
Frontier 21 Program at
Korea Institute of Science and Technology
Seoul, Republic of Korea
{mtchoi, munsang}@kist.re.kr

Abstract— **Home service robots increasingly need to provide diverse and complex services such as cooking, sweeping and dishwashing. These services inevitably require a number of software functions simultaneously. For example, the cooking service requires an arm manipulation function to grasp dishes, an navigation function to move around, an object recognition function to find foods, an speech recognition function to understand user requirements, and etc. However, when the services and software functions are executed simultaneously in a robot without run-time software management, those may cause malfunction due to resource contention. In this paper, we describe the situation that causes resource contention and formulate architecture-based adaptation in robot software systems. Based on the formulation we proposed an approach to dynamic robot software management that effectively uses robot computing resources.**

I. Introduction

Robots are incrementally replacing humans' jobs in different areas, for example, industrial robots in assembly lines, UAV(Unmanned Air Vehicles)/UGV(Unmanned Ground Vehicles) in military, and vacuum cleaner robots in houses. But these robots are only dedicated one specific service not like general purpose desktop PCs, e.g. an industrial robot assemblies only one component repeatedly for its whole life-cycle, an UAV is designed only for scouting, and a vacuum cleaner robot cannot help other chores. People, however, expect robots can assist soon our everyday life as a servant, for example, dishwashing, laundry, cooking, and sweeping. In addition, they expect the robots will enrich our life by playing or chatting with them. In other words, people want '*Home Service Robots*'.

CIR(Center for Intelligent Robots) in KIST(Korea Institute of Science and Technology) is developing home service robots for elderly people. The goal of CIR is to provide home service robots which can support chores to help handicapped elderly people and entertain them to prevent Alzheimer's disease. CIR's robot systems include diverse software functions that realize services, for example, a laser range funder based navigator and a face recognizer to support a human following service. CIR expects that the commercial version of home service robots can be released by 2015.

Unfortunately, home service robots are still on a basic stage which is unstable to provide above useful services because of two major problems: maturity and cost. Yet home service robots cannot provide stable services like typical word processors. For example, it is hard to guarantee for a robot to recognize a cup correctly every time, and to classify a refrigerator and and air conditioner by vision because of their similar shapes. However, This problem is not a software engineering issue and should be solved by improving each technologies.

Another problem is cost to build a robot. CIR's home service robot is still expensive to be a home appliance[1]. Sensors and actuators are mandatory facilities in a robot, threrfore, CIR tried to reduce computing devices. But reducing cost on computing devices inevitably leads to reducing computing power. To realize above useful services within **low cost** and **low computing power**, CIR needs to exploit computing power of a robot efficiently. We investigated how a robot uses computing power and proposed how the robot can use limited computing power effectively and efficiently.

This paper is organized as follows. Section II describes background knowledge to understand our robot software system and software architecture-based adaptation. Section III explains how to formulate dynamic software architecture management in robot software systems at run-time. In Section IV we propose an approach to dynamic software architecture management in our robot software systems by using software architecture-based adaptation at run-time. In Section V, we evaluate the effectiveness of the proposed approach. Section VI presents the conclusions.

II. Background

For more than three years, CIR is developing 'T-Rot[1]' which is a home service robot to support chores and to help elderly people. T-Rot can provide lots of services like a servant, for example, preparing a lunch, following a person, and delivering an object. Services also include

*This research was performed for the Intelligent Robotics Development Program, one of the 21st Century Frontier R&D Programs funded by the Ministry of Commerce, Industry and Energy of Korea.

†To whom all correspondence should be addressed

[1]T-Rot which is a robot being developed by CIR has full facilities such as laser range finders and arm manipulators and other necessary divices. T-Rot costs over $200,000

care and entertainment services such as checking blood pressure, playing a game and chatting with a person to take care of physical and mental status of elderly people. Each service requires a collection of software functions. For example, to serve a cup of beverage, it needs a speech recognizer(to recognize a command from a user), a dialog processor(to find out what kind of beverage the user wants), a text-to-speech module(to speak), a navigator(to move to a kitchen or to the user), and an arm manipulator(to grasp a cup). This indicates a robot must execute a set of software functions *simultaneously* and inevitably suffer from *resource contention*.

Until developing a prototype of T-Rot, CIR assumed that T-Rot starts and executes all software functions at run-time at simultaneously because they thought three or four SBCs are enough. However, CIR has developed over twelve software functions(still increasing) in the last stage of prototyping and has soon realized that it is nearly impossible to execute all software functions simultaneously because of limited resources in the robot and increasing numbers of software functions. Moreover, CIR will reduce the number of SBCs due to high cost of the robot.

Another problem is that every software function is designed and implemented to be executed in one specific SBC. Most groups of developers thought it is more efficient that the location of a software function they are developing because their software function can exploit robot computing resources independently without interference at development time and does not cause communication overhead between SBCs. But this may cause inflexibility and starvation when a set of software functions which were designed to be executed in one specific and fixed SBC is executed simultaneously. In this situation, the software functions cannot exploit enough resources(e.g. CPU time, memory and network bandwidth) due to competition, even other SBCs have enough resources. Consequently, this fact leads to malfunction of their functionalities. For example, when a collection of software functions including a laser range finder based navigator is executed in one SBC simultaneously in T-Rot, we have seen the navigator cannot move the robot successfully because the navigator cannot exploit enough CPU time from the SBC.

Moreover, every software function cannot be divided into detail and smaller modules. This problem leads to inefficient robot resource usage. We, for instance, have experienced the following situation: when a set of software functions consume 70% of CPU time of SBC-A and another set of software functions consume 80% of CPU time of SBC-B, a new software function that may consume 50% of CPU time cannot be executed even though 50% of free CPU time is available.

If every software function can be executed in any SBC and divided into smaller modules, a robot can exploit its computing resources more efficiently. But It is not applicable to examine and record all possible combinations of the locations(SBCs) of software functions to solve this problem because there are a number of combinations and the number of software function is increasing rapidly.

One possible solution to the above problems is dynamic software architecture-based adaptation[2]. This approach enables software to change its structure and behavior based on its architecture. A software architecture[3] consists of a set of components which is computing units and a set of connectors which enables communication between components. Also a software architecture defines organization of those components and connectors so that software can execute its behavior. We have adopted dynamic software architecture-based adaptation to refine software function into components and to manage(i.e. to deploy) those components efficiently and dynamically on SBCs in a robot.

The rest of this section explains structure of CIR's robot software systems that defines scope of dynamic software architecture management, and illustrates more details on dynamic software architecture-based adaptation. Based on this section, section III formulates dynamic software architecture management in home service robot software and section IV proposes how to realize dynamic software architecture management by using dynamic software architecture-based adaptation.

A. Three Layer Architecture

CIR's robot software system has a typical three layer hybrid reactive-deliberative architecture[4] which is a layered software architecture dedicated to robot systems. This architecture consists of 'the deliberative layer', 'the sequencing layer', and 'the reactive layer'. The deliberative layer understands user commands, plans a service as a task based on the commands, and recognizes the current situation based on observed data. The sequencing layer contains software functions which have not hard realtime constraints such as navigation, localization, recognition and text-to-speech. The reactive layer contains software functions called reactive modules that have hard realtime constraints.

In this research the sequencing layer is considered the target of adaptation because the deliberative layer must control the robot's behavior continuously and software functions in the reactive layer have hard realtime constraints in which dynamic adaptation may cause malfunction. Hence we concentrated to analyze and design software functions in the sequencing layer. These activities identify components and connectors and its organization(i.e. architecture). Based on these activities, robot software functions can be dynamically deployed.

B. Software Architecture-based Adaptation

Software architecture-based adaptation[2] is an approach to dynamic software evolution at run-time. This approach basically assumes that the software system which is the target of adaptation must be designed by well-defined software architecture[3]. Software architecture consists of components which execute software functionalities and connectors which connect components. A component has executable code which carries out a specific functionalities. A connector links two or more components and relays

messages between components. A software architecture organizes structure of software functions which defines connections between components and connectors.

Many researchers proposed software architecture-based adaptation approaches. Garlan proposed the Rainbow framework[5] that reconfigures the architecture of networked systems based on a modified version of Acme language[6] which can describe software architectures. Taylor proposed the C2-architecture style based[7] dynamic adaptation approach[8] which can reconfigure architectures of desktop applications designed by the C2-architecture style. In addition to above two approaches, a couple of approaches[9], [10] was examined. All the approaches provide how to organize(model) software architecture and how to implement components and connectors to be reconfigured dynamically.

Based on the examination, the process for CIR's robot software system development which enables dynamic adaptation at run-time was designed. Briefly two activities are needed; 1) architecture modeling[1] and 2) components and connectors implementation[11]. Software architectures of software functions of the robot software system were modeled by the COMET(Concurrent Object Modeling and architectural design mEThod) method already[12]. The previously examined approaches to dynamic adaptation proposed implementation methods and guidelines for each domain but not for robot software systems. Hence we have proposed the SHAGE framework[13], [11] and this research adopts its implementation guidelines.

III. Problem Formulation

As explained in section II, the robot software system needs dynamic software architecture management to handle resource contention. Before explaining the proposed approach we need to define software architecture-based adaptation in the robot software system which has limited resources. Hence we will introduce sub-architectures, components, and their relationship to help understanding.

As explained before, to support a service, T-Rot has a set of software functions such as navigation, object recognition, automatic speech recognition, and etc. In CIR's robot software system, each software function is designed as a *sub-architecture* that carries out an independent functionality. A sub-architecture defines how the functionality of sub-architecture interacts with a task(see section II-A that the deliberative layer plans and how the functionality is implemented by a set of components(i.e. composition of components). A *component* has executable code fragments to implement the (partial) functionality the sub-architecture provides and has interfaces to communicate with other components. A task needs more than one sub-architecture and a sub-architecture needs more than one component.

The problem is that every component consumes robot resources such as CPU time, memory, sensors, actuators and network bandwidth. This fact incurs each sub-architecture occupies a amount of robot resources and consequently each task needs a large portion of robot resources, sometimes, even more than entire robot resources. If a task is executed without dynamic software architecture management, it may cause malfunction as explained in section II. To handle this problem, two perspectives of dynamic software architecture management must be considered; **temporal** and **spatial** architecture management. Temporal architecture management deals with architectural evolution as a task proceeds. This management is related to prefetch. Spatial architecture management deals with architecture deployment at a specific moment. This paper only deals with spatial architecture management and leaves temporal architecture management as future work.

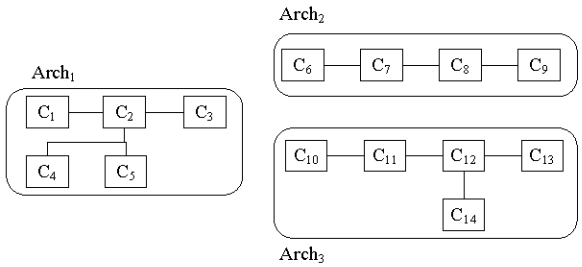

Fig. 1. Examples of sub-architectures in a robot

Spatial architecture management can be modeled by the 0-1 multidimensional, multiple knapsack problem. For example, at some moment, a task needs a set of sub-architectures when a user requests a service. Each sub-architecture requires a collection of components and these will consume robot resources. In CIR's current robot systems, one specific sub-architecture is executed in one SBC(e.g. object recognizer is executed in the vision SBC in which cameras are installed) but the task may require too many sub-architectures in one SBC and it may lead to over consumption of computing resources of the SBC. At this moment the robot needs efficient deployment of sub-architectures. Consequently, it indicates efficient deployment of components.

Let N be the number of SBCs(i.e. multiple knapsacks) and n be the number of components. Let $C = \{C_1, C_2, \cdots, C_n\}$ be a component set that will be deployed in a set of SBCs $SBC = \{SBC_1, SBC_2, \cdots, SBC_N\}$ at a moment. CPU_{SBC_i} and Mem_{SBC_i} are computing resources(i.e. multidimensional knapsacks), which the component set will use, for each SBC_i where $i = 1, 2, \cdots, N$. CPU_{C_j} and Mem_{C_j} are computing resources that each component C_j consumes where $j = 1, 2, \cdots, n$. A sub-architecture $Arch_k$ is comprised of a subset of C as depicted in figure 1 where $k = 1, 2, 3, \cdots$. Each sub-architecture defines connection between components in the sub-architecture. The following matrix depicts connection information(connectors) of sub-architecture $Arch_2$ in figure 1:

	$(C_6 - C_7)$	$(C_7 - C_8)$	$(C_8 - C_9)$
$Overhead$	1	.5	2.5

Each element in the above matrix indicates communication overhead of each connection. For example, zero means there is no connection between two components and a real numbered element larger than zero means there is a connection. Each real numbered element indicates an amount of relative communication overhead in a robot software system. These real numbered values are transformed by using system-dependant values, for example, 'MB/s'.

The goal is to deploy the component set C into the SBC set SBC under the constraint which minimizes the distribution of residue resources of SBCs as shown in figure 2. This goal is needed to absorb resource overconsumption when some components overuse resources due to some particular reasons(errors or unpredicted situations). Assuming $x_{i,j}$ has 1 if a component C_j is deployed in a SBC SBC_i and otherwise 0, the goal can be formulated as follows:

$$\text{minimize } F = \omega_1 \sum_{i=1}^{n}\sum_{j=1}^{n}(C_i - C_j) + \omega_2 V(SBC) \quad (1)$$
$$\text{where } 0 \leq \omega_1, \omega_2 \leq 1 \text{ and } 0 \leq \omega_1 + \omega_2 \leq 1$$

subject to

$$\sum_{j=1}^{n} CPU_{C_j} x_{i,j} \leq CPU_{SBC_i} \text{ for every } i = 1, 2, \cdots, n$$
$$\sum_{j=1}^{n} Mem_{C_j} x_{i,j} \leq Mem_{SBC_i} \text{ for every } i = 1, 2, \cdots, n \quad (2)$$

F in the equation (1) is the object function of this problem, $V(SBC)$ is the distribution of residue resources of SBCs, and ω_1, ω_2 are weight values of the sum of communication overhead and the distribution for each. $(C_i - C_j)$ is the communication overhead value between C_i and C_j. If we can find an matrix of $x_{i,j}$ which minimizes F and satisfies constraints in equation (2), every component can be deployed into SBCs using computing resources in the robot efficiently. The next section describes the proposed approach to obtain an possible matrix of $x_{i,j}$.

IV. Approach

This section explains processes to realize the formulated problem in section III. It needs following steps:

1) Analyzing and modeling sub-architectures,
2) Designing and implementing components in sub-architectures,
3) Evaluating resources that each component uses, and
4) Deploying components into SBCs.

The rest of this section provides brief processes to achieve the above steps.

A. Sub-architecture Analysis and Modeling

The COMET methodology[12] is adopted to construct sub-architectures of software functions in CIR's robot software systems. We have already applied the methodology to a navigator as a pilot[1]. In this methodology, each software function is analyzed by various perspectives such as static and dynamic views and modeled by UML(the Unified Modeling Language)[14].

B. Component Design and Implementation

After analyzing and modeling sub-architectures, components that constitute sub-architectures are implemented. These components should follow specific implementation guidelines proposed by our previous work[11]. By these guidelines, those components can be deployed dynamically at run-time. For example, if C_2 and C_3 are deployed in the same SBC(e.g. SBC_1), the SHAGE Framework[13] automatically connects two component by a connector which uses direct invocation. When C_3 moves to SBC_2, the framework automatically reconnect two components by a connector which uses remote invocation(e.g. RMI). This dynamic adaptation can be done by implementation guidelines that the framework provides.

C. Resource Usage Estimation

To support spatial architecture management at run-time, statistical resource usage information of every component must be estimated. Although the best way to estimate resource usage is on-line estimation that evaluates resource usage of components at run-time *after* deployment, but this way needs a lot of computation power. One alternative way is off-line estimation that evaluates resource usage at run-time *before* deployment for each sub-architecture. Even though this way cannot estimate real execution of sub-architectures, it can estimate meaningful data without overhead after deployment time. In off-line estimation, each sub-architecture is executed independently in a robot system and resource usage of each component is estimated. These estimation data of components will be used in component deployment step.

D. Component Deployment

When the task manager in the deliberative layer requests a sequence of actions to the sequencing layer, the SHAGE framework searches a set of appropriate sub-architectures. Every component in the set of sub-architectures must be deployed in SBCs to be executed. Also the deployment should not violate resource constraints in each SBC. As explained in section III this deployment problem is a 0-1 multidimensional, multiple knapsack problem. Unfortunately, finding an optimal solution of a knapsack problem is NP-complete[15]. Hence we proposed an greedy algorithm to solve the problem as follows:

1) Compute weighted sums of CPU and memory usage estimation values of every component,
2) Deploy a component from the component which consumes most resources,
3) Select a SBC to minimize the object function F, and
4) Repeat 2)~3) until all components are deployed.

For example, let a component set C has three components, C_1, C_2, C_3(their resource usage is shown in table I) and assume that there are two SBCs which can

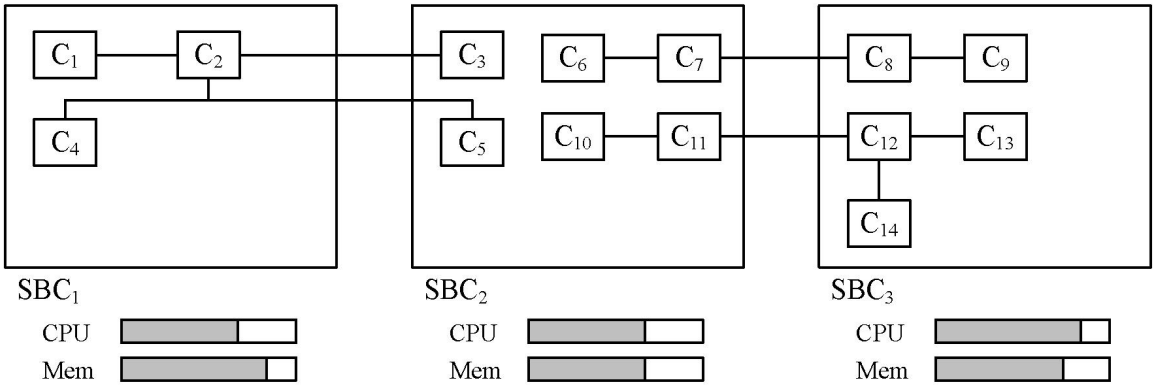

Fig. 2. An Example of component deployment

TABLE I
RESOURCE USAGE OF COMPONENTS IN SECTION IV-D

	C_1	C_2	C_3
CPU	20	40	15
Memory	5	10	4

execute those components. Suppose that weight values in the object function F are $\omega_1 = 5$ and $\omega_2 = 1$. The architecture for those components defines connections $(C_1 - C_2)$, $(C_1 - C_3)$, and $(C_2 - C_3)$ and their connection overhead data are $(C_1 - C_2) = 1.5$, $(C_1 - C_3) = 1$, and $(C_2 - C_3) = 2$. Assume that the weight of CPU usage is equal to 1 and the weight of memory usage is equal to 3. Then, calculate overall resource usage of each component is $C_1 = 35, C_2 = 70, C_3 = 27$. After calculating resource usage of components, select the component that consumes the largest resources, in this case C_2. When deploying just one component, the value of object function F is same wherever the component is deployed. Assume C_2 is deployed in SBC_1. Select C_1 as a component to be deployed in the next step. If C_1 is deployed in SBC_1, $F = 60$, and if in SBC_2, $F = 27.5$. Hence C_1 is deployed in SBC_2. In this manner C_3 is deployed in SBC_1.

Although the above greedy algorithm cannot guarantee to generate an optimal solution, but it can produce an reasonable solution in linear time. The next section gives a case study that applies the above approach to a simple situation.

V. CASE STUDY

In this section we report on a case study for evaluating dynamic software architecture management to home service robot software. In this case study, we suppose the task manager requests five software functions to achieve a specific task. Original assumption(*static deployment*) limits the location of sub-architectures of every software function. For example, all components in 'Object Recognizer' must be deployed and executed in the Vision SBC that has cameras. This assumption looked reasonable when the sub-architecture can dominate the SBC's resource. But this is not realistic because most tasks require a collection of software functions. We first verified the situation that causes malfunction when deploying sub-architectures by static deployment under the following configuration: 1) Two SBCs; one is the Vision SBC that has cameras and pan-tilt gears which controls the robot's head, the other one is called the Main SBC that has wheels, microphones, laser range finders, and speakers. 2) Each SBC is equipped with 1GB of main memory and 2.2 GHz CPU. 3) Two SBCs are connected by 100 Mbps network.

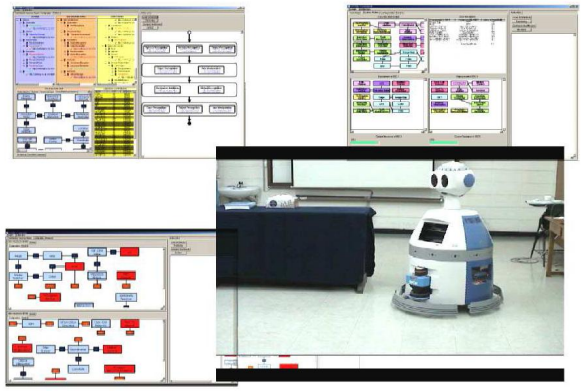

Fig. 3. Capture image of the case study in section V

Following static deployment, we have deployed a laser range finder-based navigator(has five components), a TV program recommender(four components), an arm manipulator(four components), an interaction manager(seven components), and an active audition planner(four components) into the Main SBC. At this time the Vision SBC has no resource consumption. Then, we focused on the navigator's behavior. The navigator has a constraints that checks the robot's pose and a map around the robot every 200ms to navigate safely. But under static deployment, the navigator cannot have enough computing resource(especially CPU) and cannot retrieve a pose and a map every 200ms. This causes wrong path planning and the robot cannot reach the destination. This situation can be relieved by removing more than one sub-architectures, but the task that requires

all software functions simultaneously cannot achieve its goal.

Hence, we applied our approach explained in section IV to the above situation. First, we analyzed and modeled sub-architectures of the five software functions. Then, we designed and implemented every component in the sub-architectures by using the guidelines of SHAGE framework[13], [11]. Based on the implementation, we estimated resource usage(CPU, memory, and communication overhead) of every component by profiling tools. The run-time environment of the framework adaptively deployed components of sub-architectures by using the algorithm explained in section IV when the task manager requested the five software functions. When using software architecture-based adaptation, we can find out the navigator can make a path and follow the path successfully. Figure 3 shows screen shots of user interfaces in the framework and the robot using the framework to carry out software architecture-based adaptation.

The greedy algorithm has $O(nN)$ time complexity where n is the number of components to be deployed and N is the number of SBCs. In this case study we carried out the above experiment 79 times with software architecture-based adaptation and measured time to decide the location of 24 components in two SBCs. Average time to make a decision was about 42μs(much less than one millisecond) in a SBC. This overhead doesn't influence overall performance of the robot software system. Another overhead is generated in the framework. In conventional method invocation a program calls a method directly like `foo();` but in the framework every method invocation is done by *message passing*. Message passing mechanism restricts method invocation to indirect and implicit passing of messages through a connector. Indirect and implicit mechanism may cause delays in method invocation. Hence we measured invocation overhead of both invocation mechanisms. The result of the average of direct invocation delays was 6μs and the result the average of message passing delays was 74μs on the average. Message passing takes more time by ten times but the number '**74μs**' is a quite small number that doesn't influence over system performance.

VI. Conclusions

In this paper we have described an approach to dynamic software architecture-based adaptation in robot software systems. Based on the given three layer robot architecture and software functions, this approach determines the location of software units(components) and deploys them into SBCs dynamically. To realize the approach, we have defined our robot software system, sub-architectures, and componenets. Then, we have formulated the deployment problem to a 0-1 multidimensional, multiple knapsack problem. The proposed approach has four steps; sub-architecture analysis and modeling, component design and implementation, resource usage estimation, and component deployment. In the case study, we have evaluated effectiveness of the approach and measured overhead generated by the approach.

This work suggests a number of important future directions. First is the lack of sub-architectures and components. Every participating researcher(or team) in CIR has enough technologies to implement the software function which they are responsible for and has own implementation. But most of implementations don't have well-designed architectures and components. This fact may restrict opportunities for dynamic robot software adaptation. Further study needs more effective education and reengineering researches. Second is need for more effective resource usage estimation. More precise resource usage estimation is an important technology for efficient dynamic adaptation but the proposed approach provides an estimation method before deployment time. An efficient on-line resource estimation method that has less overhead at run-time will support more effective dynamic adaptation.

References

[1] M. Kim, S. Kim, S. Park, M. Choi, M. Kim, and H. Gomaa, "Uml-based service robot software development: A case study," in *Proceedings of the 28th International Conference on Software Engineering, Shanghai*, 2006.

[2] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbingner, G. Johnson, N. Medvidovid, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, pp. 54–62, May 1999.

[3] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[4] E. Gat, "On three-layer architectures," in *Artificial Intelligence and Mobile Robots* (D. Kortenkamp, R. P. Bonnasso, and R. Murphy, eds.), MIT/AAAI, 1997.

[5] D. Garlan, S.-W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, pp. 46–54, October 2004.

[6] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," in *Foundations of Component-Based Systems* (G. T. Leavens and M. Sitaraman, eds.), ch. 3, pp. 47–67, NY: Cambridge University Press, 2000.

[7] R. N. Taylor, N. Medvidovic, K. M. Anderson, J. W. Jr., J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow, "A component- and message-based architectural style for gui software," *IEEE Transactions on Software Engineering*, vol. 22, pp. 390–406, June 1996.

[8] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *the 20th International Conference on Software Engineering*, 1998.

[9] J. Hillman and I. Warren, "An open framework for dynamic reconfiguration," in *26th International Conference on Software Engineering*, 2004.

[10] S. O. Hallsteinsen, E. Stav, and J. Floch, "Self-adaptation for everyday systems.," in *WOSS*, pp. 69–74, 2004.

[11] D. Kim and S. Park, "Designing dynamic software architecture for home service robot software," in *IFIP International Conference on Embedded and Ubiquitous Computing(EUC)* (E. Sha, S.-K. Han, C.-Z. Xu, M. H. Kim, L. T. Yang, and B. Xiao, eds.), vol. 4096, pp. 437–448, 2006.

[12] H. Gomma, *Designing Concurrent, Distributed, and Real-Time Application with UML*. Addison-Wesley, 2000.

[13] D. Kim, S. Park, Y. Jin, H. Chang, Y.-S. Park, I.-Y. Ko, K. Lee, J. Lee, Y.-C. Park, and S. Lee, "Shage: A framework for self-managed robot software," in *Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems(SEAMS)*, 2006.

[14] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 2nd ed., 2005.

[15] D. Pisinger, "An exact algorithm for large multiple knapsack problems," *European Journal of Operational Research*, vol. 114, pp. 528–541, 1999.