Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

IFAC

Software Architecture-based Approach to Self-adaptive Function for Intelligent Robots ⋆

Dongsun Kim and Sooyong Park *

* *Department of Computer Science and Engineering, Sogang University, Shinsoo-dong, Mapo-Gu, Seoul, Republic of Korea (Tel: +82-2-701-4797; e-mail: darkrsw@sogang.ac.kr).*

Abstract: An intelligent service robot helps human users with providing various services such as bringing a newspaper, recommending TV programs, and preparing meals. Each service can be accomplished by coordinating various motion actuations that are activated based on sensory data. Due to the limitation of robot computing-resources such as CPU usage and memory, the software components that implement such motion actuations can not be loaded and executed at the same time as the complexity of the service increases. That is, those components may compete with each other for the limited computing-resources, and this may result an unexpected behavior of the robot. In this paper, we propose a software architecture-based approach for self-adaptive function that optimizes the use of computing resources by supporting dynamic re-deployment of software components. Organizations of motion actuations for providing services are modeled by software architecture that describes required components and their configurations. In our approach, when a resource problem is detected, components are re-deployed across single-board computers (SBCs) in the robot while maintaining the functional and quality requirements of the components and configuration among them represented in the software architecture. We designed the self-adaptive software framework and implemented a prototype of it. We also had an experiment of our approach on an infotainment robot, and successfully proved the effectiveness of the architecture-based self-adaptive function.

1. 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 lifecycle, 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 2 describes background knowledge to understand our robot software system and software architecture-based adaptation. Section 3 explains how to formulate dynamic software architecture management in robot software systems at run-time. In Section 4 we propose an approach to dynamic software architecture management in our robot software systems by using

⋆ 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.

[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

978-1-1234-7890-2/08/$20.00 © 2008 IFAC

10.3182/20080706-5-KR-1001.4216

software architecture-based adaptation at run-time. Section 5 presents the conclusions.

2. BACKGROUND

For more than three years, CIR is developing 'T-Rot(Kim et al. [2006b])' 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 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 usage, 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 usage 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 usage of SBC-A and another set of software functions consume 80% of CPU usage of SBC-B, a new software function that may consume 50% of CPU usage cannot be executed even though 50% of free CPU usage 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(Oreizy et al. [1999]). This approach enables software to change its structure and behavior based on its architecture. A software architecture(Shaw and Garlan [1996]) 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 3 formulates dynamic software architecture management in home service robot software and section 4 proposes how to realize dynamic software architecture management by using dynamic software architecture-based adaptation.

2.1 Software Architecture-based Adaptation

Software architecture-based adaptation(Oreizy et al. [1999]) 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(Shaw and Garlan [1996]). 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(Garlan et al. [2004]) that reconfigures the architecture of networked systems based on a modified version of Acme language(Garlan et al. [2000]) which can describe software architectures. Taylor proposed the C2-architecture style based(Taylor et al. [1996]) dynamic adaptation approach(Oreizy et al. [1998]) which can reconfigure architectures of desktop applications designed by the C2-architecture style. In addition to above two approaches, a couple of approaches(Hillman and Warren [2004], Hallsteinsen et al. [2004]) 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(Kim et al. [2006b]) and 2) components and connectors implementation(Kim and Park [2006]). Software architectures of software functions of the robot software system were modeled by the COMET(Concurrent Object Modeling and architectural design mEThod) method already(Gomma [2000]). 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(Kim et al. [2006a], Kim and Park [2006]) and this research adopts its implementation guidelines.

3. PROBLEM FORMULATION

As explained in section 2, 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 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 usage, 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 2. 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.

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

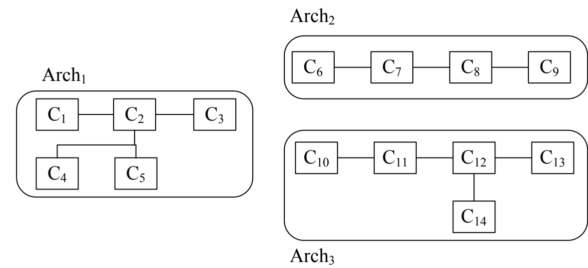

Fig. 1. Examples of sub-architectures in a robot

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:

$$\begin{array}{cccc} & (C_6 - C_7) & (C_7 - C_8) & (C_8 - C_9) \\ Overhead & 1 & .5 & 2.5 \end{array}$$

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 $x_{i,j}$.

4. APPROACH

This section explains processes to realize the formulated problem in section 3. 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.

4.1 Sub-architecture Analysis and Modeling

The COMET methodology(Gomma [2000]) 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(Kim et al. [2006b]). 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)(Booch et al. [2005]).

4.2 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(Kim and Park [2006]). 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(Kim et al. [2006a]) 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.

4.3 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.

4.4 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 3 this deployment problem is a 0-1 multidimensional, multiple knapsack problem. Unfortunately, finding an optimal solution of a knapsack problem is NP-complete(Pisinger [1999]). 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)$\sim$ 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 1) and assume that there are two SBCs which can 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.

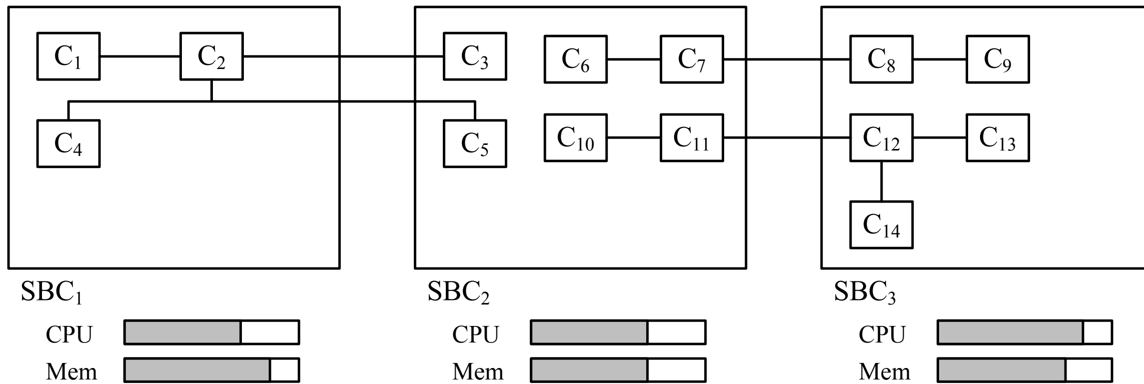

Fig. 2. An Example of component deployment

Table 1. Resource usage of components in section 4.4

	C_1	C_2	C_3
CPU	20	40	15
Memory	5	10	4

5. 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.

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

Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 2nd edition, 2005.

David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, chapter 3, pages 47–67. Cambridge University Press, NY, 2000. ISBN 0-521-77164-1.

David Garlan, Shang-Wen Cheng, An Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, October 2004.

Hassan Gomma. *Designing Concurrent, Distributed, and Real-Time Application with UML*. Addison-Wesley, 2000.

Svein O. Hallsteinsen, Erlend Stav, and Jacqueline Floch. Self-adaptation for everyday systems. In *WOSS*, pages 69–74, 2004.

Jamie Hillman and Ian Warren. An open framework for dynamic reconfiguration. In *26th International Conference on Software Engineering*, 2004.

Dongsun Kim and Sooyong Park. Designing dynamic software architecture for home service robot software. In Edwin Sha, Sung-Kook Han, Cheng-Zhong Xu, Moon Hae Kim, Laurence T. Yang, and Bin Xiao, editors, *IFIP International Conference on Embedded and Ubiquitous Computing(EUC)*, volume 4096, pages 437–448, 2006.

Dongsun Kim, Sooyong Park, Youngkyun Jin, Hyeongsoo Chang, Yu-Sik Park, In-Young Ko, Kwanwoo Lee, Junhee Lee, Yeon-Chool Park, and Sukhan Lee. Shage: A framework for self-managed robot software. In *Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems(SEAMS)*, 2006a.

Minseong Kim, Suntae Kim, Sooyong Park, MunTaek Choi, Munsang Kim, and Hassan Gomaa. Uml-based service robot software development: A case study. In *Proceedings of the 28th International Conference on Software Engineering, Shanghai*, 2006b.

Paymen Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbingner, Gregory Johnson, Nenad Medvidovid, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May 1999.

Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based runtime software evolution. In *the 20th International Conference on Software Engineering*, 1998.

D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall,

1996.

Richard N. Taylor, Nenad Medvidovic, Kenneth M. Anderson, James Whitehead Jr., Jason E. Robbins, Kari A. Nies, Peyman Oreizy, and Deborah L. Dubrow. A component- and message-based architectural style for gui software. *IEEE Transactions on Software Engineering*, 22(6):390–406, June 1996.