A Task-based Approach to Generate Optimal Software-Architecture for Intelligent Service Robots

Yu-Sik Park[1], In-Young Ko[2], and Sooyong Park[3]

[1,2]Yu-Sik Park and In-Young Ko, School of Engineering, Information and Communications University, Daejeon, Republic of Korea, e-mail : (yusikpark, iko)@icu.ac.kr
[3] Sooyong Park, Department of Computer Science, Sogang University, Seoul, Republic of Korea, e-mail : sypark@sogang.ac.kr

Abstract— **In order to provide services more reliably, intelligent service robots need to consider various factors, such as their surrounding environments, user's changing requirements, and constrained resources. Most of the intelligent service robots are controlled based on a task-based control system, which generates a task plan that consists of a sequence of actions, and executes the actions by invoking the corresponding functions. However, this task-based control system did not seriously consider resource factors even though intelligent service robots have limited resources (limited computational power, memory space, and network bandwidth). If we consider these factors during the task generation time, the complexity of the plan may become unmanageable. Therefore, in this paper, we propose a mechanism for robots to efficiently use their resources on-demand. We define reusable software-architectures corresponding to each action of a task plan, and provide a way of using the limited resources by minimizing redundant software components. We conducted an experiment of this mechanism for an infotainment robot. The experiment shows the effectiveness of our mechanism.**

I. Introduction

An intelligent service robot needs to support robustness of continually providing services for users at run-time. To accomplish this, many researchers have developed techniques for robots to deal with changing tasks in the real world [1, 2]. They developed robot control systems to generate a plan for a task. The control system generates a plan for robots to move from an initial state to a goal state. The plan consists of actions that the robot needs to perform to accomplish the task. Such plan is normally represented in a DAG-based structure.

Task-based robots generally use three-layer software architecture as depicted in Fig. 1. The Decision Layer generates a task plan and supervises the execution of the plan. The Execution Layer is in charge of an execution of the actual functions and controls for the plan. The Function Layer provides basic functions such as perception and navigation functions [3].

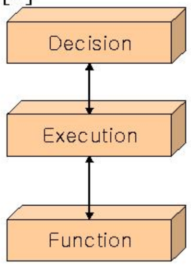


Fig. 1 Three-layer software architecture for robots

However, this traditional architecture has three limitations as follows:

- In this three-layer architecture, as we add more conditional factors to consider, the complexity of the plan generation becomes unmanageable. If the number of factors increases, the depth of the plan graph grows exponentially. This would tremendously increase the planning efforts [4].
- This architecture does not suggest any way to efficiently use the limited resources of the robots by maintaining only the essential functions that are required to perform a task.
- The decision layer decides statically which functions to use rather than it chooses the functions dynamically based on environmental condition.

To deal with these problems, the SHAGE framework has been developed. The SHAGE framework enables robots to analyze generated task plans and select appropriate software architectures for the actions in a plan. The framework also provides a mechanism to reduce resource consumption by identifying and removing redundant functionalities in the selected architectures. By using this mechanism, we could successfully reduce the complexity of the plan generation, and overcome the limited resources of robots.

The rest of the paper is organized as follows. In Section 2, we briefly explain the SHAGE framework. In Section 3, we elaborate on the task-based architecture generation mechanism. Section 4 presents the experiment that we conducted. Section 5 discuses about related works that handle resource variability issues. We draw a conclusion and present future works in Section 6.

II. The SHAGE framework

SHAGE is a framework to support self-managed software for intelligent service robots, stands for Self-Healing, Adaptive, and Growing software [5]. Our previous research aimed to adapt robot software to various situations by using run-time software architecture and a component reconfiguration mechanism. However, our purpose in this paper is to provide software system that helps intelligent service robots provide services without a hitch.

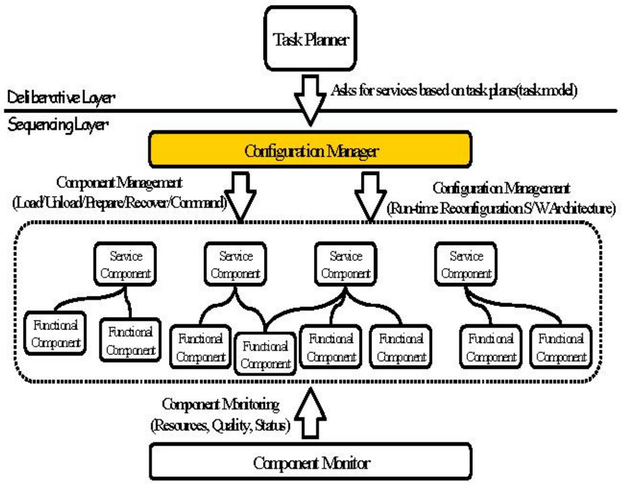

Task Planner
Deliberative Layer
Sequencing Layer
Asks for services based on task plans(task model)
Configuration Manager
Component Management (Load/Unload/Prepare/Recover/Command)
Configuration Management (Run-time Reconfiguration S/W Architecture)
Service Component
Functional Component
Component Monitoring (Resources, Quality, Status)
Component Monitor

Fig. 2 Role of the SHAGE framework

The SHAGE framework is located between the decision layer (deliberative layer) and the execution layer (sequencing layer). The framework plays a role of a configuration manager as depicted in Fig. 2. The configuration manager has two roles: component management and configuration management. Component management is to manage components at run-time by loading and unloading components for a plan. Configuration management is to generate and reconfigure software architecture instances. In realizing this framework, the positioner is added to our framework. The positioner is responsible for assigning selected components to SBCs (Single Board Computers). This paper focuses on selecting and merging sub-architectures for a task plan.

III. Task-based architecture generation

In order to efficiently use limited resources of robots, it is important to minimize redundant software components that provide similar functionality. Therefore, we propose a task-based approach to reduce the resource consumption by identifying redundant components in robot software architectures. We define a reusable sub-architecture corresponding to each action of a plan.

The architecture generation phase consists of three steps: sub-architecture search, architecture consolidation, and concrete architecture generation as depicted in Fig. 3. In the sub-architecture search phase, the given task plan is analyzed and sub-architectures are selected. In the architecture consolidation phase, the selected sub-architectures are merged into a single architecture by collapsing similar functionalities. Finally, component instances are selected in the concrete architecture generation phase.

The sub-architecture search and architecture consolidation phases deal with abstract architectures. Search and inference in these steps are based on an ontology-based representation. In this section, we explain the ontology that we constructed for these phases, and the detail steps of the phases.

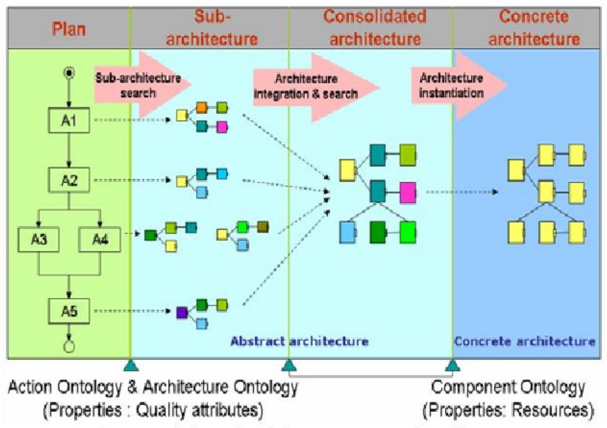

Plan
Sub-architecture
Consolidated architecture
Concrete architecture
Sub-architecture search
Architecture integration & search
Architecture instantiation
A1
A2
A3
A4
A5
Abstract architecture
Concrete architecture
Action Ontology & Architecture Ontology (Properties : Quality attributes)
Component Ontology (Properties: Resources)

Fig. 3 Task-based architecture generation phase

A. Ontology description model

There are three main ontologies that we use in the architecture generation phases: the 'Action', 'Sub-architecture', and 'Components' ontologies as shown in Fig. 4. The Action ontology describes services which robots can provide. Navigation, object recognition, and speech recognition are example services in this ontology.

The Sub-architecture ontology has information about the sub-architectures that provide functionality to support an action. Each class of the Sub-architecture ontology includes a description about which actions are supported by the sub-architecture. This enables robots to infer appropriate sub-architectures for an action in a plan. In addition, this ontology keeps detail architecture descriptions in a property. This architecture description specifies components and connectors that are needed for the architecture, and relations among them. The components specified in the architecture description are associated with the classes in the Component ontology (see Section 3.2 for details).

The 'Component' ontology is constructed based on the functionality of components. This ontology is used to select component instances that are required to concretize a consolidated architecture. This ontology also allows robots to identify alternative components that provide similar functions. In addition, the Component ontology includes descriptions of resources constraints such as computational power, memory space, and network bandwidth of components. By using this information, we can choose a set of component for an architecture such that the components optimize the resource consumption.

B. Sub-architecture search

A task plan consists of a sequence of actions and conditional statements. To make robots operate according to the generated plan, actions in the task plan need to be identified and their corresponding sub-architectures need to be selected. A task plan from the decision layer specifies necessary actions to perform. Sub-architectures for a specific action can be selected by using the Sub-architecture ontology. There can be multiple sub-architectures that support an action. Among the candidate sub-architectures, the robot selects a specific one that satisfies quality requirements of a user and minimizes the resource consumption.

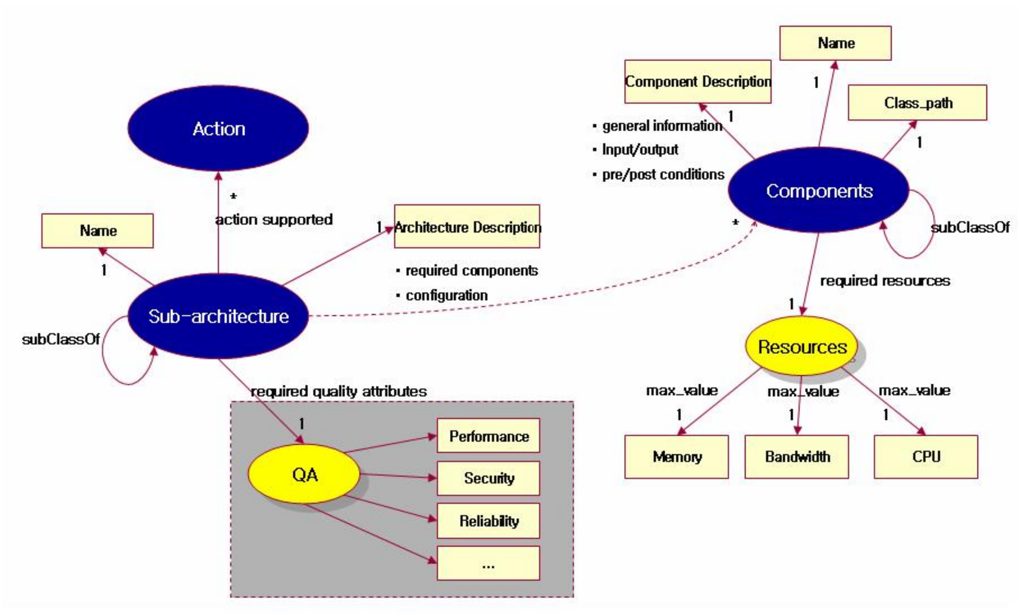


Fig. 4 Ontology description model

A sub-architecture is an abstract architecture corresponding to an action. It specifies a set of abstract components and their connections. That is, a sub-architecture specifies functionality of components and relations among the constituent components. At the architecture consolidation phase explained in Section III. D, specific component instances are selected for the abstract components in a sub-architecture. In this way, multiple implementations of a sub-architecture can be produced by selecting different sets of component instances.

Usually, some actions are used in multiple task plans. Therefore, by defining a sub-architecture for an action, we can maximize the reusability of the sub-architecture, and the components that constitute the sub-architecture.

C. Consolidated architecture generation

The selected sub-architectures in the previous phase need to be merged while reducing redundant components that provide similar functionalities. The hierarchy of the Component ontology is used to identify similarity between component functionalities. Component functionalities in a higher level in the Component ontology subsume their lower-level functionalities. In other words, higher-level functionalities are more general than the lower-level functionalities in the ontology hierarchy. Similar components in different sub-architectures can be identified by using the following relations:

- *Equivalence relation*: components in different sub-architectures correspond to the same class in the Component ontology. In other words, components provide the same functionality.
- *Subsumption relation*: one component in a sub-architecture provides more specific functionality than a component in another sub-architecture. In other words, they appear in the same parent-child path in the Component ontology hierarchy.

For example, when two sub-architectures are selected as shown in Fig. 5, the component 'c_{12}' of the sub-architecture 1 and the component 'c_{23}' of the sub-architecture 2 can be merged because 'c_{12}' is equivalent to 'c_{23}', and 'c_{11}' of the sub-architecture 1 and 'c_{22}' of the sub-architecture 2 can be merged because 'c_{11}' subsumes 'c_{23}'.

D. Concrete architecture generation

Since a consolidated architecture is an abstract architecture, component instances need to be selected and associated to the architecture to be executed. Components are selected based on expected resource consumption. The expected resource consumption of a component is described in the 'resource' property of the Component ontology. Expected resource consumption of a component is evaluated by using off-line simulation. The 'positioner' of the SHAGE framework monitors a resource status of SBCs and decides components to use among the selected candidate components. The 'positioner' also deploys the selected components to SBCs with considering the resource status of the SBCs and the total resource consumption of the selected components. We will not explain the detail of this component- deployment issue in this paper.

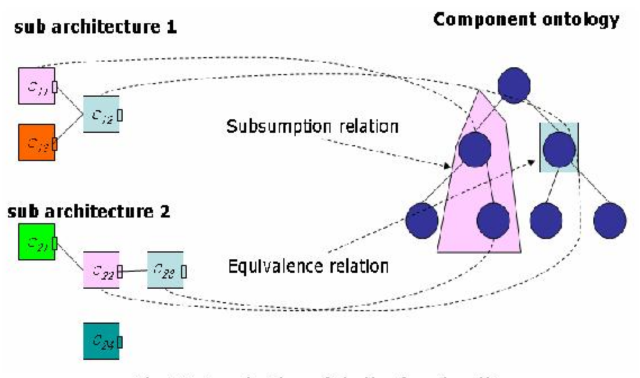


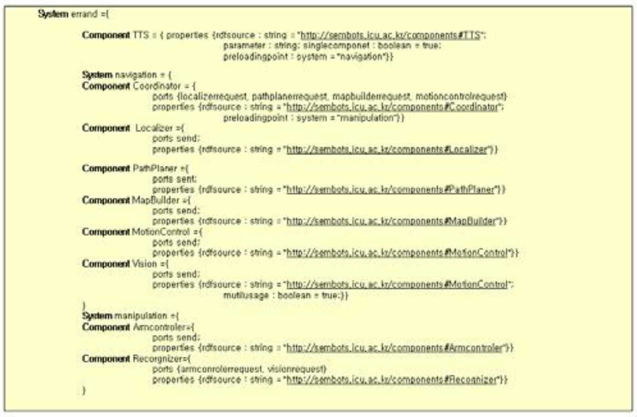
Fig 5 Determination of similar functionality

IV. Implementation

In this section, we explain how sub-architectures are represented formally, and describe the implementation details of the architecture broker and the component broker.

Sub-architectures are described in Acme, which is a simple, generic architecture description language that can be used to specify structure and properties of software architecture [6]. We use Acme because it provides powerful way of specifying various properties of architectural elements.

Fig. 6 is an example of sub-architecture description for a navigation function. The navigation sub-architecture consists of 'Coordinator', 'Localizer', 'PathPlanner', 'MapBuilder', and 'MotionControl'. The functionalities of those components are specified in the property construct. This property specifies the URI of a component-ontology class for its corresponding functionality.



```
System errand ={

        Component TTS = { properties {rdfsource : string = "http://sembots.icu.ac.kr/components#TTS";
                                     parameter : string; singlecomponet : boolean = true;
                                     preloadingpoint : system = "navigation"}}

        System navigation = {
        Component Coordinator = {
                    ports {localizerrequest, pathplanerrequest, mapbuilderrequest, motioncontrolrequest}
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#Coordinator";
                                     preloadingpoint : system = "manipulation"}}
        Component  Localizer ={
                    ports send;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#Localizer"}}

        Component PathPlaner ={
                    ports sent;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#PathPlaner"}}
        Component MapBuilder ={
                    ports send;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#MapBuilder"}}
        Component MotionControl ={
                    ports send;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#MotionControl"}}
        Component Vision ={
                    ports send;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#MotionControl";
                                     mutilusage : boolean = true;}}
        }
        System manipulation ={
        Component Armcontroler={
                    ports send;
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#Armcontroler"}}
        Component Recorgnizer={
                    ports {armconrolerrequest, visionrequest}
                    properties {rdfsource : string = "http://sembots.icu.ac.kr/components#Recognizer"}}
        }
```

Fig. 6 An example of sub-architecture description

Fig. 7 shows the architecture of the architecture and the component broker. When a task plan is given from the decision layer, the message handler finds necessary actions and then sends them to the architecture brokering engine. The architecture brokering engine then accesses ontologies stored in the repository to search and consolidate sub-architectures. The brokering engines are implemented in Java, and use the Jena library which provides APIs to access and manage ontology-based models [7].

Finally, the component brokering engine searches candidate components for a selected sub-architecture. Information about the selected sub-architecture and required components is packaged by the service information generator and sent to the configuration manager in the SHAGE framework. The visualization unit graphically presents brokering stages and architecture consolidation process. Fig. 9 shows the screen shot of the visualization unit.

The ontologies are created by using the Protégé ontology editor [8]. The onologies are described in RDF (Resource Description Framework) as depicted in Fig. 8. RDF is a framework for representing semantic information on the Semantic Web. RDF descriptions of sub-architectures and components are stored in a repository system [9].

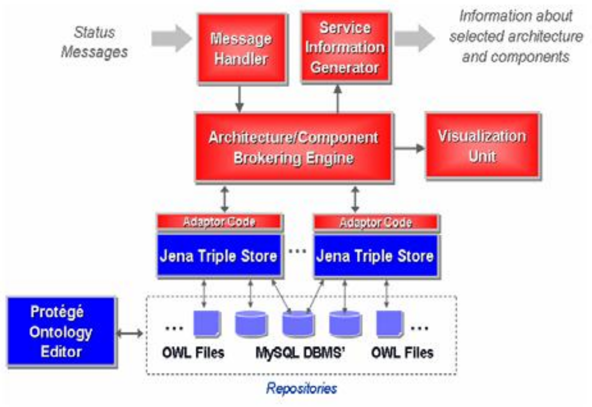


Fig. 7 Broker architecture

The entire process of task-based architecture generation can be demonstrated through the visualization unit of the brokers as shown in Fig. 9. The GUI consists of three views: the ontology view (the upper side), architecture view (the lower-left side), and candidate component view (the lower-right side). The ontology view presents an action that is currently focused, and the selected sub-architectures, and their constituent components. The architecture view shows a graphical structure of a sub-architecture and its architecture description in Acme. The candidate component view shows the resource properties of the candidate components for the consolidated architecture.

V. Related works

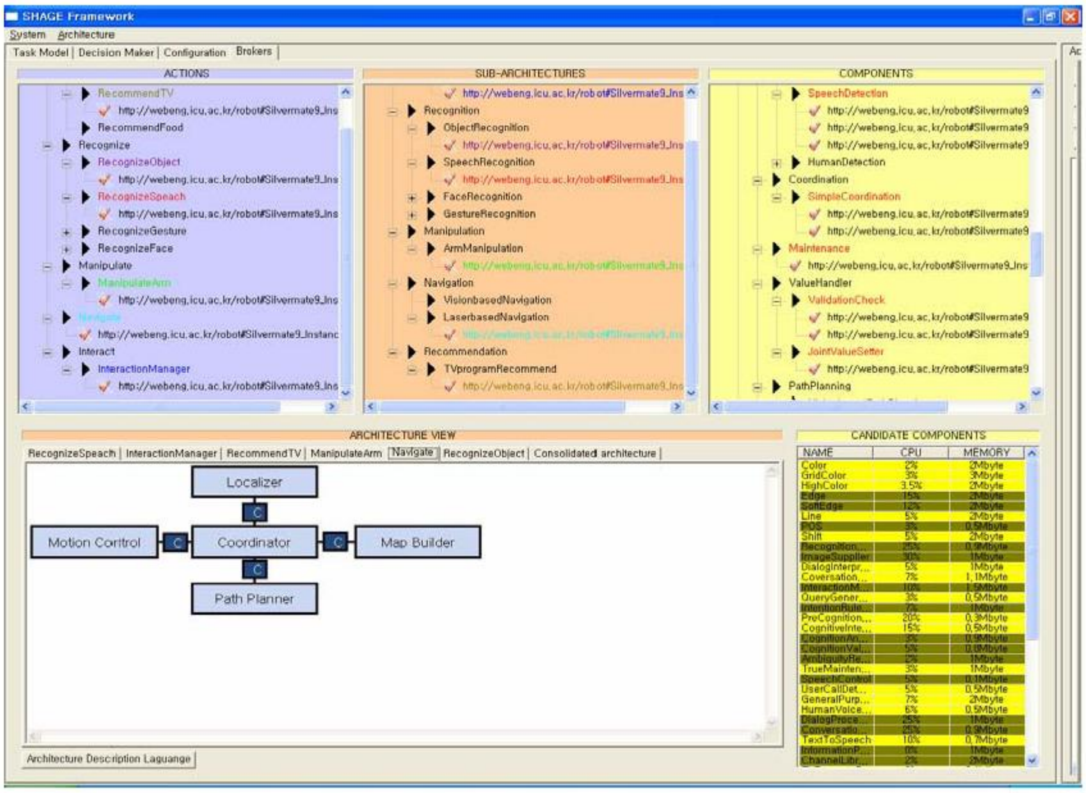
Ubiquitous computing environment includes similar factors to consider as we do for intelligent service robots. The Aura project developed a self-adaptive computing infrastructure that automates the configuration and reconfiguration of heterogeneous computing environments [10]. Aura provides users with services that match required service qualities by doing optimal resource allocation.

Jie, et al. at the Microsoft research institute proposed an approach to share intermediate sensing and computing results among tasks [11]. In this research, they experimented with network devices called Microserver that accept users' tasks. Microserver is constrained by CPU speed, memory size, and communication bandwidth. In this research, they identified overlapping operations from multiple tasks and shared intermediate results among tasks.

The above two researches are similar to our research in terms of a task-based approach to handle resource variability. However, while they regard applications as a reconfigurable unit, we consider components and sub-architectures as reconfiguration units.

```
<rdf:Description rdf:about="&robot;Silvermate9_Instance_20015"
        robot:CLASSPATH="slam.stereovision.visionbasedslam"
        robot:COMPONENT-DESCRIPTION-
        PATH="SLAM.StereoVision.VisionbasedSLAM.xml"
        robot:COMPONENT-NAME="SLAM"
        rdfs:label="Silvermate9_Instance_20015">
<rdf:type rdf:resource="&robot;VisionbasedMapBuilding"/>
<rdf:type rdf:resource="&robot;VisionbasedLocalization"/>
</rdf:Description>
```

Fig. 8 An example of a component description written in RDF


SHAGE Framework
System Architecture
Task Model | Decision Maker | Configuration Brokers |
ACTIONS
RecommendTV
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
RecommendFood
Recognize
RecognizeObject
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
RecognizeSpeach
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
RecognizeGesture
RecognizeFace
Manipulate
ManipulateArm
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
http://webeng.icu.ac.kr/robot#Silvermate9_Instanc
Interact
InteractionManager
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
SUB-ARCHITECTURES
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
Recognition
ObjectRecognition
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
SpeechRecognition
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
FaceRecognition
GestureRecognition
Manipulation
ArmManipulation
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
Navigation
VisionbasedNavigation
LaserbasedNavigation
Recommendation
TVprogramRecommend
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
COMPONENTS
SpeechDetection
http://webeng.icu.ac.kr/robot#Silvermate9
http://webeng.icu.ac.kr/robot#Silvermate9
http://webeng.icu.ac.kr/robot#Silvermate9
HumanDetection
Coordination
SimpleCoordination
http://webeng.icu.ac.kr/robot#Silvermate9
http://webeng.icu.ac.kr/robot#Silvermate9
Maintenance
http://webeng.icu.ac.kr/robot#Silvermate9_Ins
ValueHandler
ValidationCheck
http://webeng.icu.ac.kr/robot#Silvermate9
http://webeng.icu.ac.kr/robot#Silvermate9
JointValueSetter
http://webeng.icu.ac.kr/robot#Silvermate9
PathPlanning
ARCHITECTURE VIEW
RecognizeSpeach | InteractionManager | RecommendTV | ManipulateArm | Navigate | RecognizeObject | Consolidated architecture |
Localizer
C
Motion Control
C
Coordinator
C
Map Builder
C
Path Planner
Architecture Description Laguange
CANDIDATE COMPONENTS

NAME	CPU	MEMORY
Color	2%	2Mbyte
GridColor	3%	3Mbyte
HighColor	3.5%	2Mbyte
Edge	15%	2Mbyte
SoftEdge	12%	2Mbyte
Line	5%	2Mbyte
POS	3%	0.5Mbyte
Shift	5%	2Mbyte
Recognition...	25%	0.9Mbyte
ImageSupplier	30%	1Mbyte
DialogInterpr...	5%	1Mbyte
Coversation...	7%	1.1Mbyte
InteractionM...	10%	1.5Mbyte
QueryGener...	3%	0.5Mbyte
IntentionRule...	7%	1Mbyte
PreCognition...	20%	0.3Mbyte
CognitiveInte...	15%	0.5Mbyte
CognitionAn...	3%	0.9Mbyte
CognitionVal...	5%	0.8Mbyte
AmbiguityRe...	2%	1Mbyte
TrueMainten...	3%	1Mbyte
SpeechControl	5%	0.1Mbyte
UserCallDet...	5%	0.5Mbyte
GeneralPurp...	7%	2Mbyte
HumanVoice...	6%	0.5Mbyte
DialogProce...	25%	1Mbyte
Conversatio...	25%	0.9Mbyte
TextToSpeech	10%	0.7Mbyte
InformationP...	0%	1Mbyte
ChannelLibr...	2%	2Mbyte

Fig. 9 GUI of the brokers

In addition to resource constraints, Volpe, et al. at California Institute of Technology proposed a two-layer architecture for robotics to make existing components for robots be reused [12]. In this research, they decreased the complexity of component development for robots by using object-oriented design in a function layer and by making abstracted components provide extensibility of systems. This research aims to enable application developers to efficiently develop new components by using the existing ones. When components for robots are developed under this environment, it enables robot to select and acquire various existing components.

VI. Conclusion and Future Work

In this paper we described a task-based approach to generate optimal software-architecture for intelligent service robots. To make intelligent service robots reliably provide their services with the limited resources, we have developed an architecture broker and a component broker. Our approach includes an architecture generation mechanism that selects appropriate sub-architectures and components for the actions specified in a task plan. We defined reusable sub-architectures corresponding to each action of a task plan, and proposed a way to select sub-architectures and components to produce concrete architectures. Intelligent service robots, even though their resources are limited, need to provide various services for the users. Separating resource factors from the task plan generation reduces the complexity of task plan generation dramatically.

However, in addition to the topological optimization that we presented in this paper, we are currently working on developing a temporal optimization method. It will provide the capability of preloading necessary components that are needed in a temporal window of a plan. Also, it will help robots decide when components are no longer needed and need to be unloaded. This temporal optimization will be possible by analyzing the temporal relation between actions of a task plan.

VII. Acknowledgments

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.

VIII. References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An Architecture for Autonomy, The International Journal of Robotics Research, Vol. 17, No. 4, 315-337.

[2] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. Bonnasso, and R. Murphy, editors, Artificial Intelligence and Mobile Robots, Boston, MA, 1998. MIT Press.

[3] ChuXin Chen and Mohan M. Trivedi, Task Planning and Action Coordination in Integrated Sensor-Based Robots, IEEE Transaction on SYSTEMS, MAN, AND CYBERNETICS, Vol. 25, No. 4, APRIL 1995.

[4] Jayaputera, G.; Loke, S.; Zaslavsky, A., Performance evaluation of dynamically assembled multiagent systems, Intelligent Agent Technology, IEEE/WIC/ACM International Conference on, 19-22 Sept. 2005 Page(s):451 – 454.

[5] Dongsun Kim, Sooyong Park, Youngkyun Jin, Yu-Sik Park, In-Young Ko, Kwanwoo Lee, Junhee Lee, Yeon-Chool Park, Sukhan Lee. SHAGE: A Framework for self-managed Robot Software, International Conference on Software Engineering, Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, Shangai, China.

[6] David Garlan, Robert Monroe, David Wile, Acme: Architectural Description of Component-Based Systems, Proceedings of the 1997

conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada.

[7] Jena Semantic Web Framework, http://jena.sourceforge.net

[8] What is Protégé?, http://protege.stanford.edu/overview

[9] Hyung-Min Koo, In-Young Ko. A Repository Framework for Self-Growing Robot Software, Proceedings of 12th Asia-Pacific Software Engineering Conference (APSEC'2005), Taiwan, December 2005.

[10] João Pedro Sousa, Vahe Poladian, David Garlan, Bradley Schmerl, Mary Shaw, Task-based Adaptation for Ubiquitous Computing, IEEE Transactions on Systems, Man and Cybernetics, Vol. 36, 328-340, May 2006.

[11] Jie Liu, Elaine Cheong, and Feng Zhao, "semantics-based optimization across uncoordinated tasks in networked embedded systems" Proceedings of the 5th ACM Conference on Embedded Software (EMSOFT 2005), September 18-22, 2005, Jersey City, New Jersey, USA.

[12] R. Volpe, et al., "The CLARAty Architecture forRobotic Autonomy." Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 10-17 2001.