SHAGE: A Framework for Self-managed Robot Software[*]

Dongsun Kim, Sooyong Park[†]
{darkrsw, sypark}@sogang.ac.kr

Youngkyun Jin, Hyeongsoo Chang
{jiny1u, hschang}@sogang.ac.kr

Department of Computer Science, Sogang University, Mapo-Gu, Seoul, Republic of Korea

Yu-Sik Park, In-Young Ko
Information and Communications University
Yuseong-gu, Deajeon, Republic of Korea
{yusikpark, iko}@icu.ac.kr

Kwanwoo Lee
Hansung University
Sungbuk-gu, Seoul, Republic of Korea
kwlee@hansung.ac.kr

Junhee Lee, Yeon-Chool Park,
Sukhan Lee
Sungkyunkwan University, Suwon, Gyeonggi-do,
Republic of Korea
leejunhee@ece.skku.ac.kr,
fearhope@gmail.com,
lsh@ece.skku.ac.kr

ABSTRACT

Behavioral, situational and environmental changes in complex software, such as robot software, cannot be completely captured in software design. To handle this dynamism, self-managed software enables its services dynamically adapted to various situations by reconfiguring its software architecture during run-time. We have developed a practical framework, called SHAGE (Self-Healing, Adaptive, and Growing SoftwarE), to support self-managed software for intelligent service robots. The SAHGE framework is composed of six main elements: a situation monitor to identify internal and external conditions of a software system, ontology-based models to describe architecture and components, brokers to find appropriate architectural reconfiguration patterns and components for a situation, a reconfigurator to actually change the architecture based on the selected reconfiguration pattern and components, a decision maker/learner to find the optimal solution of reconfiguring software architecture for a situation, and repositories to effectively manage and share architectural reconfiguration patterns, components, and problem solving strategies. We conducted an experiment of applying the framework to an infotainment robot. The result of the experiment shows the practicality and usefulness of the framework for the intelligent service robots.

*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

Categories and Subject Descriptors

D.2.11 [**Software Architecture**]: Patterns

General Terms

Design

Keywords

Self-managed Software, Software Architecture, Robot Software

1. INTRODUCTION

To deal with dynamic behavior, situations and environmental changes at run-time, current software engineering practices are not adequate due to the hardness of software modification. An approach to resolve this problem could be making software really "*soft*" that enables run-time software modification. Even though the current paradigm for software development aims at making software more modifiable. Especially in embedded systems such as robots, there is a lot of needs for changing its functionalities at run-time, even though there is no software architecture in it.

This research is currently being conducted as part of the 'intelligent service robot for the elderly' project in the Center for Intelligent Robotics (CIR) at the Korea Institute of Science and Technology (KIST). This project was faced with 'how to satisfy changing requirements more faster at run-time'. For example, navigation is one of the most important

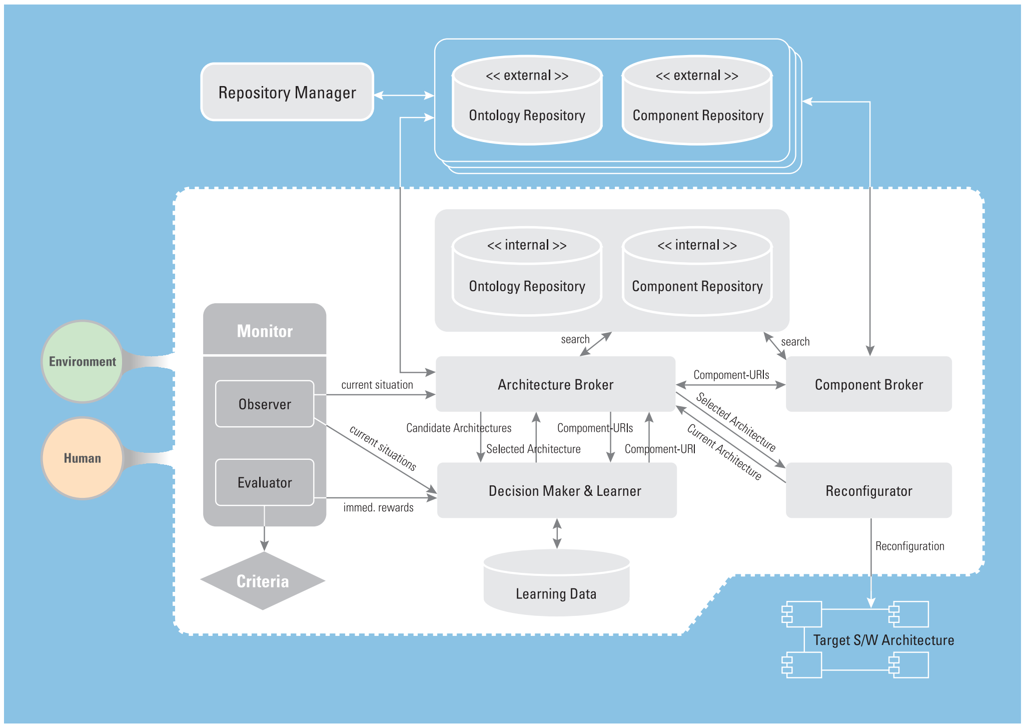


Figure 1: SHAGE Framework consists of two parts; the inner part has seven modules: the Monitor(not in our scope yet), the Architecture Broker, the Component Broker, the Decision Maker, the Learner, the Reconfigurator, and (internal) Repositories. The outer part consists of repository servers that provide repository services

functions in robot systems. Usually, the user of a robot commands to move with only a goal position. But situations can be diverse; a) when the user held a party, there may be many persons. They are unrecorded and moving objects for the robot. In this case, the robot needs to move more carefully, even though it may be getting slow. b) when the user is home alone, there are only recorded objects. In this case, the robot can move more quickly. In both cases a) and b), the user only put the goal position to the robot and would not give more details such as ‘move carefully’, ‘there’s lots of persons’, ‘I don’t care a bit of collisions’ and so on(of course, the user can notice ‘what’s the problem’ when the robot encounters some unidentifiable situations but basically, he wouldn’t give details of an operation before the robot executes the operation). Thus, the robot must recognizes situations of the environment and infers the user’s requirements to adapt its behavior. In addition to adaptation, the robot should not repeat previous experience which returns bad rewards. To do this, the robot must learn and memorize situations.

In this paper, we propose a framework to implement self-managed software in Section 2. In Section 3, we show the results of a case study conducted in a robot. Then, we draw conclusion in Section 4.

2. SHAGE FRAMEWORK

2.1 Framework Overview

SHAGE[1] Framework is developed to give a self-managed software capability to robot software in the project. The Framework consists of two parts which are separated by a dashed line as depicted in figure 1. The inner part is installed each robot and consists of seven modules. The outer part provides repository services for robots to obtain new knowledge which describes ‘how to adapt’. The environment and the user are not part of the framework but the framework continuously communicates with them to decide ‘when to adapt’ and ‘how to adapt’. The target architecture represents the software architecture of the robot and the target of adaptation.

Seven modules in the inner part of the framework are the Monitor(not in our scope yet), the Architecture Broker, the Component Broker, the Decision Maker, the Learner, the Reconfigurator, and the Internal Repositories. The monitor is responsible for observing the current situation of the environment(this is what observer does) and evaluating the result of adaptation that the framework does(this is what evaluator does). The architecture broker searches candidate architectures based on architecture reconfiguration strategies and composes candidate component compositions for the selected architecture by using concrete component retrieved by the component broker. The component broker finds concrete components which will be arranged into an architecture and retrieves the components from repositories. The decision maker determines an best-so-far architecture from a set of candidate architectures that the architecture broker found and an best-so-far component composition from candidate component compositions that the architecture broker composed. The learner accumulates rewards evaluated by the

[1]formerly it was AlchemistJ as described in [1]

evaluator in the monitor and the decision maker uses these accumulated rewards to choose the best-so-far architecture and the best-so-far component composition. The learning data is a knowledge repository for the learner. The reconfigurator manages and reconfigures the software architecture of the robot based on the best-so-far architecture and the best-so-far component composition which are selected by the decision maker. The internal repositories consists of the ontology repository and the component repository. The ontology repository contains architecture reconfiguration strategies that describes functionalities the robot must have and component ontologies that describes characteristics of a component. The component repository contains components implemented to be used in the robot software architecture.

The outer part of the framework is a set of servers containing external repositories. Each server has an ontology repository and a component repository as the inner part has. Internal repositories in the robot requests new ontologies and components when the robot cannot adapt its behavior to the current situation properly. Also External repositories broadcasts new knowledge to update robots' internal repositories globally. The repository manager is installed in each server and has tools for addition and removal of ontologies and components.

The framework is activated when the monitor detects a new situation from the environment or receives a new requirement from the user. When the observer in the monitor detects a situation which needs software architectural adaptation, the monitor issues adaptation with the observed current situation to the architecture broker(startAdaptation(ISituation s)) as described in figure 4. The architecture broker requests the current architecture of the robot(getCurrentArch()) to search a set of suitable architecture reconfiguration strategies from the internal ontology repository(searchArch()). Then, the architecture broker asks the decision maker to select the most suitable software architecture of the robot(called an best-so-far architecture) based on the set of architecture reconfiguration strategies(selectArch(ArchSet set)). Once the decision maker returns an best-so-far architecture, the architecture broker requests a set of suitable components for the architecture from the internal component repository(getComponentSet()). The component broker retrieves binary code and information of components from (internal/external) component repositories(getComponent()). After receiving components from the component broker, the architecture broker composes compositions by using the retrieved components to fill the selected architecture and asks an best-so-far component composition to the decision maker. Then, the architecture broker passes the selected architecture and the selected component composition to the reconfigurator(ReconfigurArch()). The reconfigurator adds, removes, and changes components of the current software architecture and makes connections between components. When the reconfigurator reports completion of reconfiguration, the architecture broker reports that 'adaptation is done' to the monitor. Then, the evaluator in the monitor begins to examine the behavior of the robot for a while. If the monitor detects the next situation the evaluator passes rewards of the behavior to the learner. The learner accumulates rewards for the decision maker to use afterwards.

The above cycle is the process of the framework. The next sections describes each modules in the framework in detail.

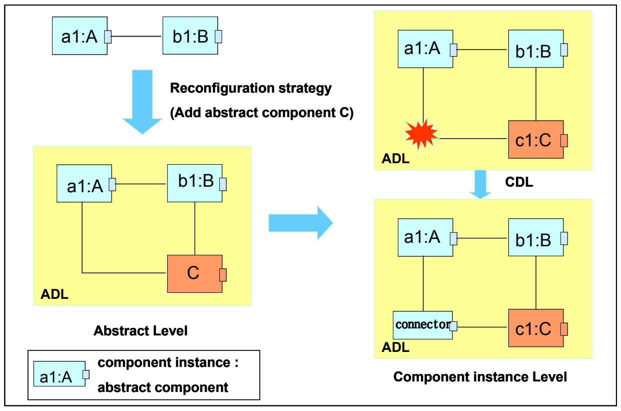


Figure 2: Architecture reconfiguration in an abstract level and component level

2.2 Architecture Broker

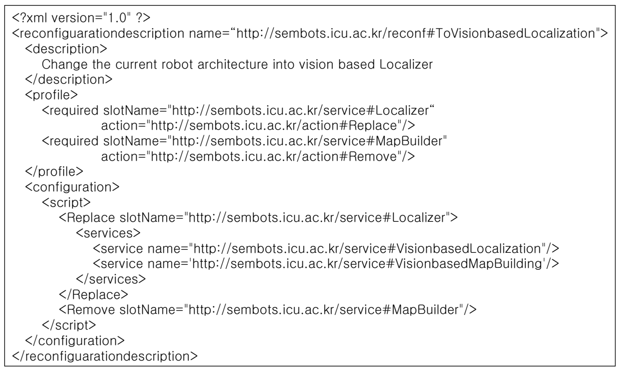
The architecture broker finds appropriate architecture-reconfiguration strategies when a robot needs to reconfigure its software architecture to overcome a problematic situation. The robot may need to add, remove or replace a component of its software architecture to provide a capability that can handle the problem. The software reconfiguration process is composed of two phases as shown in figure 2. While the architecture broker is in charge of software reconfiguration in an abstract level, the reconfigurator actually reconfigures the robot software architecture based on the abstract reconfiguration strategy and by using the components selected by the component broker (see Section 2.3). The purpose of,fig:brokering making the software reconfiguration in two different levels is to make all possible candidate components be evaluated and utilized (will be explained in detail in Section 2.5).

The architecture broker searches architecture reconfiguration strategies based on the situation detected by the monitor. A reconfiguration strategy is described in XML as shown in figure 3. An XML-based architecture-reconfiguration description includes the URI of an abstract component to be changed, and the required functionality (specified by a URI that represents a functional ontology) of a new component to be added. Based on this reconfiguration strategy, the architecture broker requests a set of component instances that provide the required functionality to the component broker. The architecture broker finally delivers the selected components with the reconfiguration operations to the reconfigurator.

2.3 Component Broker

The component broker searches appropriate software components for solving a situation. In our previous research [2, 3], the component broker infers candidates software components based on the situation, strategy, and component ontologies. The role of the component broker has been extended to searching components that fit to a target software architecture.

The following are the steps to find a component by the component broker. At first, upon the architecture broker's request on necessary components for reconfiguring software architecture, the component broker searches for the candi-



```
<?xml version="1.0" ?>
<reconfiguarationdescription name="http://sembots.icu.ac.kr/reconf#ToVisionbasedLocalization">
  <description>
     Change the current robot architecture into vision based Localizer
  </description>
  <profile>
     <required slotName="http://sembots.icu.ac.kr/service#Localizer"
               action="http://sembots.icu.ac.kr/action#Replace"/>
     <required slotName="http://sembots.icu.ac.kr/service#MapBuilder"
               action="http://sembots.icu.ac.kr/action#Remove"/>
  </profile>
  <configuration>
     <script>
        <Replace slotName="http://sembots.icu.ac.kr/service#Localizer">
           <services>
              <service name="http://sembots.icu.ac.kr/service#VisionbasedLocalization"/>
              <service name='http://sembots.icu.ac.kr/service#VisionbasedMapBuilding'/>
           </services>
        </Replace>
        <Remove slotName="http://sembots.icu.ac.kr/service#MapBuilder"/>
     </script>
  </configuration>
</reconfiguarationdescription>
```

Figure 3: An example of an architecture reconfiguration description

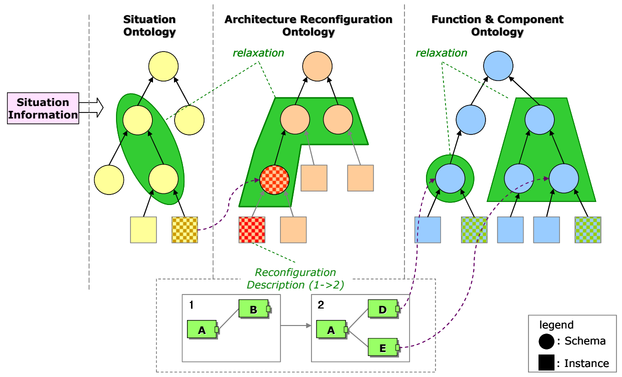

Situation Ontology
Architecture Reconfiguration Ontology
Function & Component Ontology
Situation Information
relaxation
relaxation
Reconfiguration Description (1->2)
1
B
A
2
D
A
E
legend
●: Schema
■: Instance

Figure 5: Architecture/component brokering processes

date components that provide the functionality specified in the architecture reconfiguration description. If the component broker cannot find any candidate components in the internal component repository, or the decision maker decides that there is no appropriate component among the selected candidates, the component broker requests the component acquisition engine to search and collect components from external component repositories (see Section 2.6 for details).

The brokering processes of the architecture broker and the component broker are depicted in figure 5.

1. Based on the situation information received from the monitor, a situation ontology is inferred.

2. The architecture broker searches an architecture-reconfiguration description (defined as an ontology) that is associated with the situation ontology.

3. The architecture broker extracts an architecture-reconfiguration procedure from the selected architecture-reconfiguration description.

4. The architecture broker comprehends the required functionality of the components included in the architecture-reconfiguration description, and requests them to the component broker.

5. The component broker infers appropriate components based on their functionality represented in a function ontology.

2.4 Decision Maker & Learner

To decide and learn which software component is suitable for the current situation, we employed a variant of CBDT(Case-based Decision Theory) [4] introduced by Gilboa and Schmeidler [5].

CBDT is motivated by the ideas from the well-known *case-based reasoning* (CBR) [6] and *reinforcement learning* [7] in the machine learning literature; the "similar" problems have "similar" solutions and a learner can improve his learning by the feedback of his decision based on trial-and-error performance evaluation. CBDT chooses an action (i.e., software component configuration) based on the *performance* of potential actions in previous problems that are *similar* to the current one.

CBDT suffers from the cases where the values of similarity measure between problems are pretty low. To get away with this problem, the idea of "satisficing" (satisfice = satisfy + sacrifice) decision making [8][4] has been introduced. A threshold is employed and the decision maker explores the actions in $\mathcal{A}$ if the linear combination of the benefits experienced so far for the best action is lower than the threshold until it finds an action that gives a bigger value than the threshold. Still, even with the threshold strategy, CBDT does not guarantee that the selected action is an optimal action for the current problem.

We incorporated ϵ-greedy exploration-exploitation strategy from reinforcement learning into CBDT to tackle down the sub-optimality problem. If the value of the weighted sum of the best action is lower than the threshold, an action in $\mathcal{A}$ that have not been tried so far is selected uniformly. If not, with probability ϵ, an action in the set of actions in $\mathcal{A}$ that have not been tried so far is selected uniformly (exploration) and with probability $1-\epsilon$, the decision maker takes the current best action (exploitation).

2.5 Reconfigurator

The reconfigurator manages and re-organizes the software architecture of the robot without suspension at run-time. Reconfiguration of the software architecture is based on the selected architecture reconfiguration strategies and the selected component composition which are selected by the decision maker. To enable run-time reconfiguration, SHAGE framework defines some rules for designing software architecture.

Software architecture is defined by two levels; the abstract level and the concrete level. At the abstract level, there are only *slots*. A slot represents an abstract component that describes services. A service describes functionality that a slot should provide. A service does not indicates a specific method or a concrete component but describes what messages a slot can be requested and what results a slot returns. An architecture reconfiguration strategy only deals with reconfiguration of the abstract level as shown in figure 3.

At the concrete level, each slot is filled by a concrete component. A concrete component is an executable code, for example .class files in Java, implemented by predefined component implementation rules. Every component in the framework must implement common interfaces containing messages such as 'start', 'stop', and 'suspend' and specific interfaces containing messages that the component can receive and give. These specific interfaces are described by the com-

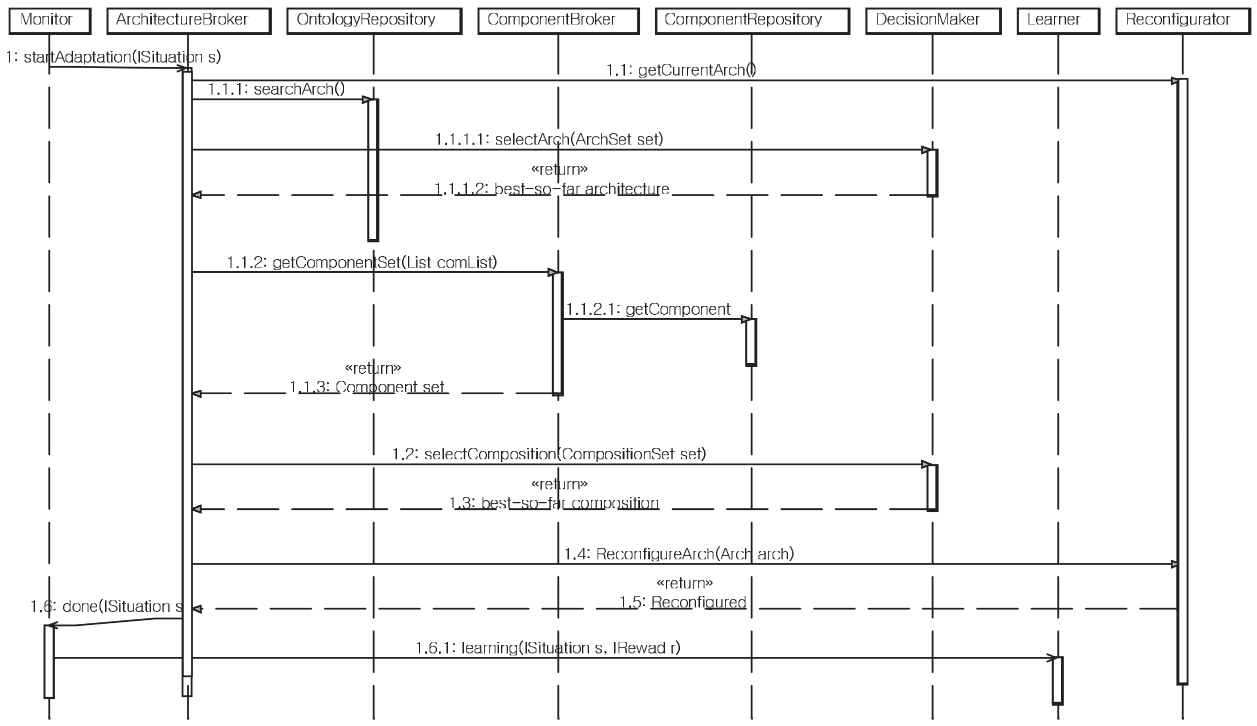

Monitor
ArchitectureBroker
OntologyRepository
ComponentBroker
ComponentRepository
DecisionMaker
Learner
Reconfigurator
1: startAdaptation(ISituation s)
1.1: getCurrentArch()
1.1.1: searchArch()
1.1.1.1: selectArch(ArchSet set)
«return»
1.1.1.2: best-so-far architecture
1.1.2: getComponentSet(List comList)
1.1.2.1: getComponent
«return»
1.1.3: Component set
1.2: selectComposition(CompositionSet set)
«return»
1.3: best-so-far composition
1.4: ReconfigureArch(Arch arch)
«return»
1.5: Reconfigured
1.6: done(ISituation s)
1.6.1: learning(ISituation s, IRewad r)

Figure 4: The process of SHAGE Framework is composed of interactions mainly between the inner part of the framework

```
<?xml version="1.0" encoding="euc-kr"?>
<Component>
   <name>Navigator.Mapbuilder:LaserbasedMapbuilder</name>
   <description>Laser sensor-based mapbuilder</description>
   <thread value="false"/>
   <language value="CPP"/>
   <deployment value="MainSBC"/>
   <location URI="navigator.mapbuilder.laserbasedmapbuilder.LaserbasedMapbuilder"/>
   <provided-interfaces>
        <service type="Algorithmic.MapBuilding.LaserbasedMapBuilding"
   name="MapBuilder">
                  <msg name='ReadMap'>
                           <reponse>
                                    <arg name='Map'
   type='Primitive.double[500][500]'/>
                           </reponse>
                  </msg>
                  <msg name='UpdateMap'>
                           <arg name='dRobotPos' type='Primitive.double[3]'/>
                           <reponse>
                                    <arg name='Map'
   type='Primitive.double[500][500]'/>
                           </reponse>
                  </msg>
        </service>
   </provided-interfaces>
   <required-interfaces>
   </required-interfaces>
</Component>
```

Figure 6: An example of component description

ponent description language shown in figure 6. The component description language is a XML-based language and describes required interfaces that contains messages that the component can request to other components and provided interfaces that contains messages that the component can offer to other components.

When the architecture broker requests reconfiguration of the robot software architecture, the reconfigurator re-organizes the architecture based on the selected architecture reconfiguration strategy and places components based on the selected component composition and places connectors between components based on the component descriptions. The reconfigurator measures mismatches between components and finds suitable connectors. For example, two components have been deployed in two different machines, the reconfigurator selects a remote connector which enables remote communication such as RPC or RMI. After all components are connected, the reconfigurator sends a 'start' messages to every new component in the architecture and reports that reconfiguration is done to the architecture broker.

2.6 Repositories

The repository system for storing and managing architecture-reconfiguration descriptions and components is composed of four main elements: a component repository, an ontology repository, an external acquisition engine, and a component retirement plan[9, 10]. The component repository stores the physical components that are available in a domain. The ontology repository stores and manages the three types of ontology explained in Section 2.3. The external acquisition engine contacts external repository systems and acquires ontologies and components that are needed to solve a problem that couldn't be solved by using internal components. The component retirement plan is for deciding the components in the internal repository to be removed or updated based on the usage history and component revision information received from external repository systems.

Figure 7 shows the repository architecture of our framework. The external acquisition engine receives a request from the architecture broker or the component broker, searches and collects architecture reconfiguration strategies or components from external repositories, and stores them into the internal repository.

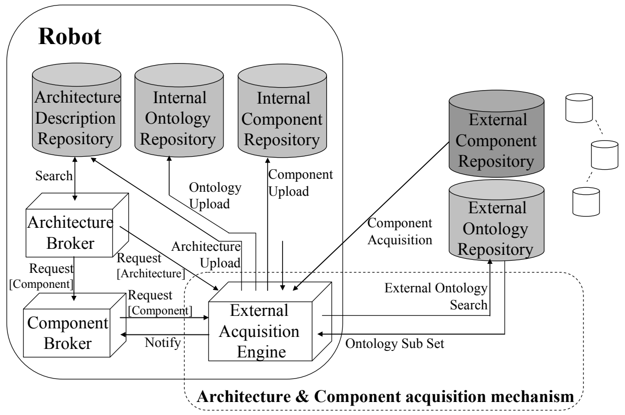


Figure 7: Repository architecture for self-growing robot software

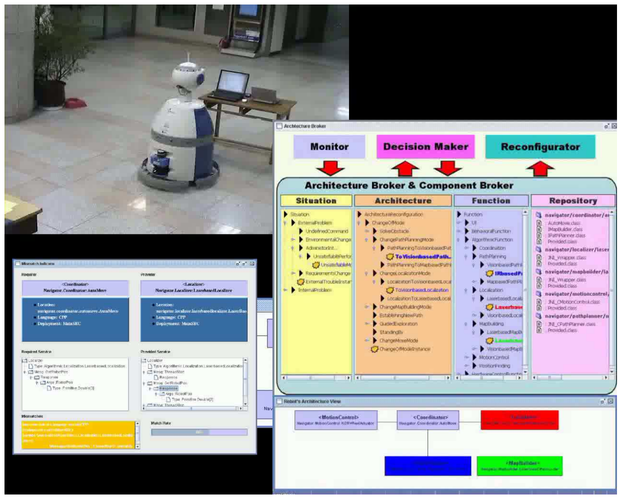


Figure 8: A captured image of the experiment

3. EXPERIMENT

We conducted an experiment to show effectiveness of the framework on a robot. The robot is a prototype of service robots and named 'infotainment robot'. This robot has two SBC(Single Board Computer)s; one is called 'Main SBC' and another is called 'Vision SBC'. These two SBCs are connected by an 100Mbps line. The Main SBC communicates with 'two laser sensors(front and rear)', 'two IR sensors(front and rear)', and 'two wheels(left and right)' through RS-232C. The Vision SBC is dedicated for getting visual data through two stereo vision cameras.

The experiment was designed as follows: 1. initially the user of the robot needs 'more faster navigation', so the robot is configured mainly to use faster sensors(say, laser sensors). 2. while moving, the robot is stuck by a table because the bottom of the table is empty but the current sensors can detect only knee height objects. Then, the user asks the robot to move 'more carefully'. 3. The robot tries to reconfigure its software architecture to detect objects that the current sensors cannot detect. We implemented a few components and configured the initial software architecture of the robot for navigation as shown in figure 9.(a). Each component can be executed independently. 'MotionControl' component controls wheels and 'Localizer' measures the current position of the robot based on encoder data which means how many degrees the wheels rotated. 'MapBuilder' makes a map around the robot based on sensor data from laser sensors. 'PathPlanner' plans a path from the current position to a goal position based on data from 'Localizer' and 'MapBuilder'. 'Coordinator' gets the goal position from the user of the robot and relays data between components. Based on these components the robot can moves without collisions except tables.

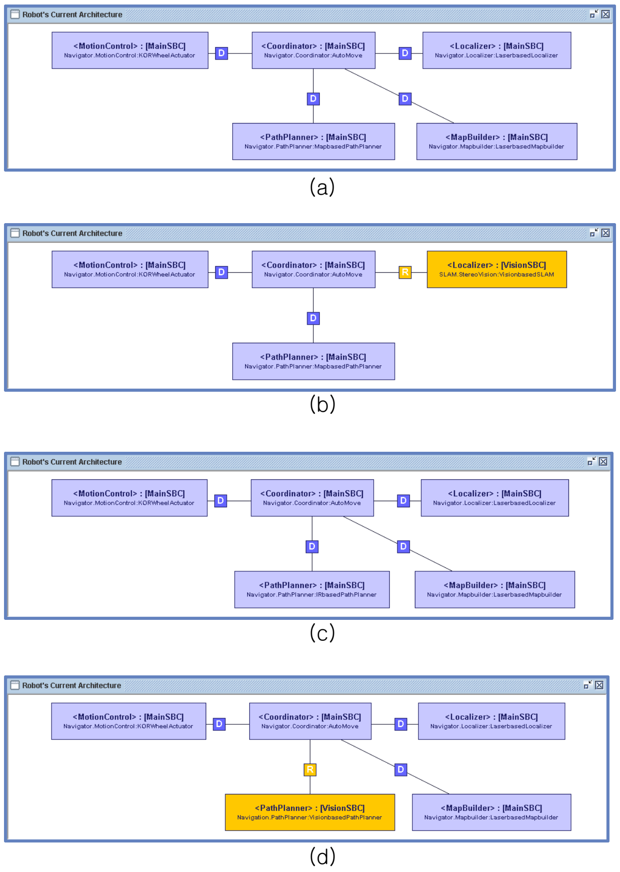
We designed a room as an experimental environment. we placed two tables; one was covered by a tablecloth and the other was not(shown in figure 8). The robot was placed at the same line on which the two tables were placed. In other words, "robot - table with a cloth - table without a cloth" on the same line in sequence. After configuring the initial architecture of the robot, The user put a goal position to 'Coordinator' and requested 'more faster maneuver'. The goal position was between two tables and the robot verified that the current architecture was suitable for the user's requirement. The robot decided the initial architecture is enough because laser sensors were very fast and precise. Then, the robot started to move and its laser sensors could detect a tablecloth, so the robot could move without collisions. This was not an abnormal situation, so the robot did not need adaptation. Then, we put the other goal position over the second table. In this case, the robot could not detect the table and rushed to the table. The user requested the robot to stop and to realize the situation and to adapt it.

When the robot was requested to adapt its behavior, the monitor in the framework detected the current situation. This situation was passed to the architecture broker. The architecture broker began the adaptation process as described in sector 2.1. The situation was the current architecture could not detect all object in the room so the robot needed to find other functionalities to detect some other objects which could not be observable with the current architecture. The framework tried to apply architecture configurations including various components such as SLAM(Simultaneous Localization And Mapping) component(figure 9.(b)), IR-based Path Planner Component(figure 9.(c)) and etc. But the framework realized(and learned) these was not able to solve the situation, and finally selected the architecture including 'vision-based path planner' component(figure 9.(d)). After this experiment, we initialized the robot position and repeated the experiment. The robot learned the situation and could overcome the situation without trial-and-error.

This experiment shows the framework enables the robot to adapt its behavior by reconfiguring its software architecture. During (a)-(d) the user did not interrupt and put more inputs. Although the monitor was not implemented and feedback was specified by the user significantly, adaptation process after feedback was done automatically.

4. CONCLUSIONS

In this paper, we proposed the SHAGE framework as self-managed software for the intelligent service robot domain. SHAGE offers main features to support self-managed software at run-time: the situation monitor (not in our scope yet), the architecture/component brokers, the decision maker/learner, the reconfigurator, and the repositories. The framework observes the situation of the surrounding environment of a robot and searches possible architecture



Robot's Current Architecture
<MotionControl> : [MainSBC]
Navigator.MotionControl:KORWheelActuator
<Coordinator> : [MainSBC]
Navigator.Coordinator:AutoMove
<Localizer> : [MainSBC]
Navigator.Localizer:LaserbasedLocalizer
<PathPlanner> : [MainSBC]
Navigator.PathPlanner:MapbasedPathPlanner
<MapBuilder> : [MainSBC]
Navigator.Mapbuilder:LaserbasedMapbuilder
D
(a)
Robot's Current Architecture
<MotionControl> : [MainSBC]
Navigator.MotionControl:KORWheelActuator
<Coordinator> : [MainSBC]
Navigator.Coordinator:AutoMove
<Localizer> : [VisionSBC]
SLAM.StereoVision:VisionbasedSLAM
<PathPlanner> : [MainSBC]
Navigator.PathPlanner:MapbasedPathPlanner
D
R
(b)
Robot's Current Architecture
<MotionControl> : [MainSBC]
Navigator.MotionControl:KORWheelActuator
<Coordinator> : [MainSBC]
Navigator.Coordinator:AutoMove
<Localizer> : [MainSBC]
Navigator.Localizer:LaserbasedLocalizer
<PathPlanner> : [MainSBC]
Navigator.PathPlanner:IRbasedPathPlanner
<MapBuilder> : [MainSBC]
Navigator.Mapbuilder:LaserbasedMapbuilder
D
(c)
Robot's Current Architecture
<MotionControl> : [MainSBC]
Navigator.MotionControl:KORWheelActuator
<Coordinator> : [MainSBC]
Navigator.Coordinator:AutoMove
<Localizer> : [MainSBC]
Navigator.Localizer:LaserbasedLocalizer
<PathPlanner> : [VisionSBC]
Navigation.PathPlanner:VisionbasedPathPlanner
<MapBuilder> : [MainSBC]
Navigator.Mapbuilder:LaserbasedMapbuilder
D
R
(d)

Figure 9: Various software architectures of the robot during adaptation

reconfiguration strategies and component compositions that are suitable for handling the situation faced. The framework also allows the robot software system to choose the best-so-far architecture reconfiguration strategy and component composition during run-time based on previous experiences.

In order to verify the practicality of the framework, we have adapted the SHAGE framework to an infotainment robot and examined the adaptation capability of the robot software. By reconfiguring the architecture, the infotainment robot could successfully adapt its behavior (supported based on software architecture) to an exceptional situation, and continue its task.

5. ACKNOWLEDGMENTS

The authors would like to thank Jeihun Lee at Sungkyunkwan University, and Hyung-Min Koo and Ki-Hyeon Kim at Information and Communications University (ICU) for their helps in conducting the experiments in the present paper.

6. REFERENCES

[1] D. Kim and S. Park, "Alchemistj: A framework for self-adaptive software," in *The 2005 IFIP International Conference on Embedded And Ubiquitous Computing (EUC'2005), LNCS3824*, pp. 98–109, December 2005.

[2] H. Lee, H. Shin, I. Y. Ko, , and H. J. Choi, "A semantically-based component selection mechanism for robot software," in *2005 Korean Conference on Software Engineering*, 2005.

[3] H. Lee, H. J. Choi, and I. Y. Ko, "A semantically-based component selection mechanism for intelligent service robots," in *4th Mexican International Conference on Artificial Intelligence*, 2005.

[4] I. Gilboa and D. Schmeidler, "Case-based decision theory," *Quarterly Journal of Economics*, vol. 110, pp. 605–639, 8 1995.

[5] I. Gilboa and D. Schmeidler, "Case-based optimization," *Games and Economic Behavior*, vol. 15, pp. 1–26, 1996.

[6] J. Kolodner, *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[8] J. G. Marc and H. A. Simon, *Organizations*. Blackwell Publishers, 1993.

[9] H.-M. Koo and I.-Y. Ko, "A repository framework for self-growing robot software," in *Proceedings of 12th Asia-Pacific Software Engineering Conference (APSEC2005), Taiwan*, 2005.

[10] H.-M. Koo and I.-Y. Ko, "A component repository framework for self-growing robot software," in *the 32nd KISS Fall Conference*, 2005.