서비스 로봇을 위한 Self-Managed 소프트웨어 프레임워크 개발†

서강대학교 박수용 · 장형수 · 김동선
한국정보통신대학교 고인영
성균관대학교 박연출
한성대학교 이관우

1. 서 론

로봇과 같은 내장형 소프트웨어는 동적으로 변화하는 환경의 상황과 사용자의 요구사항을 실행시간에 소프트웨어를 멈추지 않고 반영해야 한다. 그러나 현재 소프트웨어에 적용되고 있는 소프트웨어 공학 방법론은, 동적 수정에 대한 고려를 하지 않았기 때문에 이것들을 반영하기에는 부적절하다. 이것을 가능케 하기 위해서는 실행시간에도 소프트웨어를 수정할 수 있도록 'run-time softness'를 구현하는 접근 방법이 필요하다. 현재의 소프트웨어 공학 방법론들이 설계시간에 소프트웨어 변경을 좀 더 쉽게 할 수 있도록 하고 있음에도 불구하고, 점점 사용자들은 실행시간에 실행을 멈추지 않고 소프트웨어가 변경될 수 있도록 요구하고 있다. 특히 내장형 소프트웨어 시스템의 경우, 다양하게 환경의 상황이 변화하고 사용자의 요구사항도 지속적으로 변화하고 있지만, 대부분은 소프트웨어 아키텍처의 구성조차 고려되지 않고 있다.

본 연구는 KIST의 지능 로봇 센터(CIR: Center for Intelligent Robotics)의 노인 생활 지원을 위한 서비스 로봇 개발 프로젝트의 일환으로 시작되었다. 이 연구는 '어떻게 실행시간에 변화하는 사용자의 요구사항을 실행을 멈추지 않고 로봇 소프트웨어에 좀 더 빠르게 반영할 수 있을까?'라는 질문에서부터 시작되었다. 예를 들면, 주행 기능은 로봇의 중요 기능중 하나이다. 보통 사용자는 로봇에게 단순히 목표 지점만 알려주고 이동하라고 명령한다. 그러나 이 때 구체적인 상황은 다양할 수 있다: a) 만약 사용자가 파티를 열고 있다면 집안에는 등록되지 않고, 이동하는 장애물인 손님이 많이 있을 것이다. 이런 경우 로봇은 속도를 늦추더라도 최대한 충돌을 없애는 방법으로 주행해야 할 것이다. b) 만약 사용자가 집에 혼자 있을 경우 이미 등록된 장애물만 있을 것이고 이때는 장애물 식별은 간단하게 할 수 있고 좀더 빠르게 주행할 수 있을 것이다. a)와 b)의 경우 모두 일반적으로 사용자는 목표 지점만 입력해줄 뿐, 더 자세한 요구사항, 예를 들면, '조심해서 주행해라', '집안에 사람이 많다', '좀 부딪히는건 신경안쓰니 빨리 주행해라' 등을 말해주지 않는다(물론 사용자가 로봇이 어디엔가 부딪혔을 때 무엇이 문제라는 것을 알려줄 수는 있다. 그러나 사용자는 로봇이 어떤 기능을 수행하기 전에 미리 모든 자세한 요구사항을 말하지 않는다). **로봇은 자신의 행위를 적절히 적응시키기 위해서 주변의 상황과 사용자의 요구사항을 추론해야만 한다. 또한 로봇은 이전에 했던 실수를 반복하지 말아야 하고 이를 위해서는 학습 기능을 갖고 있어야 한다.**

2장에서는 self-managed 소프트웨어 구현을 위한 프레임워크를 제안하고 3장에서는 사례 연구 결과를 보여준다. 마지막으로 4장에서는 본 논문의 결론을 제시한다.

2. SHAGE 프레임워크

2.1 개요

SHAGE(Self-Healing, Adaptive and Growing SoftwarE)[1)] 프레임워크는 로봇 소프트웨어에 self-managed 기능을 부여하기 위해 개발되었다. 프레임워크는 그림 1에서 나와 있듯이 크게 두 부분으로 나뉘어 진다. 점선 안쪽 부분은 로봇내부에 설치되는데 7개의 모듈로 구성되어 있다 점선 바깥쪽 부분은 로봇이 어떻게 적응해야 하는지에 대한 새로운 지식을 저장하기 위한 저장소 서비스를 제공한다. 로봇 외부의 환경(Environment)과 사용자(User)는 프레임워크의 한 부분은 아니지만 프레임워크는 지속적으로 이들과 상호작용하면서 '언제, 어떻게 적용해야 하는지'를 결정

† 이 연구(논문)는 산업자원부 지원으로 수행하는 21세기 프론티어 연구개발사업(인간기능 생활지원 지능로봇 기술개발사업)의 일환으로 수행되었습니다.

1) 이전에는 AlchemistJ[1]로 불리었다.

한다. 대상 아키텍처(target architecture)는 실제로 로봇의 기능을 담당하는 소프트웨어를 말한다.

프레임워크 내부의 7개의 모듈은 Monitor(아직 연구의 범위는 아니다), Architecture Broker, Component Broker, Decision Maker, Learner, Reconfigurator, 내부 저장소(repository)들을 말한다. Monitor는 현재 환경의 상황(Monitor 내부의 Observer가 하는 일이다)을 관찰하고 프레임워크가 수행한 적응 행위의 결과를 평가(Evaluator가 하는 일이다)한다. Architecture Broker는 아키텍처 재구성 전략(architecture reconfiguration strategy)들을 검색하여 현재 상황에 적용가능한 후보 아키텍처들을 추출한다. 그리고 후보 아키텍처 중 결정된 아키텍처에 적용될 후보 컴포넌트 구성(candidate component composition)들을 생성한다. 이 후보 컴포넌트 구성은 Component Broker가 내/외부 컴포넌트 저장소에서 가져온 Concrete Component를 기반으로 한다. Decision Maker는 Architecture Broker가 준 후보 아키텍처 중 현재 상황에 최적의 아키텍처를 Learner가 축적한 정보에 기반하여 결정하다. 또한 후보 컴포넌트 구성 중 현재 상황과 앞서 선택한 최적 아키텍처에 가장 알맞은 컴포넌트 구성을 선택한다. Learner는 Evaluator로부터 들어온 현재 적응행위에 대한 평가 결과를 축적하여 나중에 Decision Maker가 아키텍처와 컴포넌트 구성을 선택하는데 이용하게 한다. Reconfigurator는 현재 실행되고 있는 로봇 소프트웨어의 아키텍처를 관리하고 적응이 요청되었을 때 결정된 아키텍처와 컴포넌트 구성을 기초로 아키텍처를 재구성한다. 내부에 있는 저장소 중 Ontology Repository는 아키텍처 재구성 전략들과 컴포넌트 온톨로지들을 저장한다. Component Repository는 실행가능한 컴포넌트 코드를 저장한다.

프레임워크의 외부는 저장소 서비스를 제공하는 서버들의 집합으로 구성된다. 각 서버는 Ontology Repository와 Component Repository를 갖고 있다. 외부의 저장소들은 로봇이 내부 저장소에 있는 온톨로지나 컴포넌트만으로는 현재 상황에 적절하게 적응할 수 없을 때 새로운 온톨로지나 컴포넌트를 제공한다. Repository Manager는 각 서버에 설치되어 새로운 온톨로지나 컴포넌트를 추가하거나 기존의 온톨로지나 컴포넌트를 관리하는 도구를 제공한다.

프레임워크는 Monitor가 외부 환경으로부터 새로운 상황을 인지하거나 사용자로부터 새로운 요구사항을 받았을 때부터 시작된다. Observer가 소프트웨어 아키텍처가 변경되어야하는 상황을 인식하면 Architecture Broker에게 현재 상황에 맞게 적응할 것을 요청한다

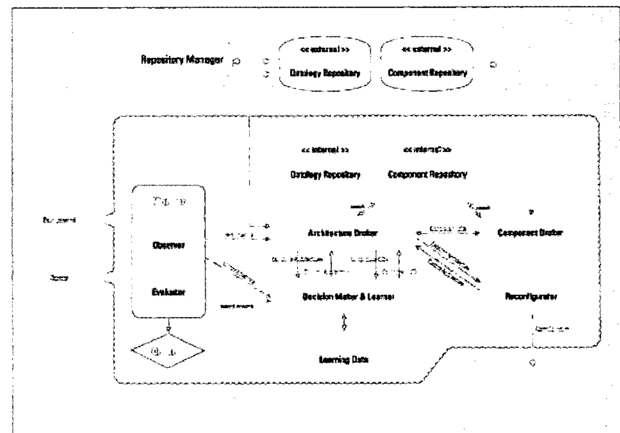


그림 1 SHAGE 프레임워크는 크게 두 부분으로 나뉘어진다; 내부 구조는 다시 7개의 모듈로 다음과 같은 구성된다: Monitor, Architecture Broker, Component Broker, Decision Maker & Learner, Reconfigurator, Ontology Repository, Component Repository. 외부 구조는 저장소 서비스를 제공하는 서버들로 구성된다.

(그림 1의 startAdaptation()). Architecture Broker는 현재 아키텍처를 Reconfigurator에게 요청하고(getCurrentArch()), 이것을 기반으로 후보 아키텍처 재구성 전략을 내부 Ontology Repository로부터 검색(searchArch())한다. Architecture Broker는 후보 아키텍처 재구성 전략 중 가장 현재 상황에 맞는 아키텍처 재구성 전략을 Decision Maker에게 선택(selectArch())하도록 한다. 선택된 아키텍처에 적용한 컴포넌트를 선택하기 위해 Component Broker에 적절한 컴포넌트를 요청한다(getComponentSet()). Component Broker는 컴포넌트 저장소로부터 컴포넌트를 검색하여 가져온다(getComponent()). 컴포넌트를 받은 후에 Architecture Broker는 선택된 아키텍처에 적용될 수 있는 컴포넌트의 조합을 여러 개 구성하여 그 중 가장 적절한 조합을 선택하도록 Decision Maker에 요청한다(selectComposition()). 아키텍처와 아키텍처를 구성할 컴포넌트가 선택되면, Reconfigurator에게 아키텍처의 재구성을 요청한다(ReconfigureArch()). Reconfigurator는 컴포넌트를 추가/삭제/교체하여 현재 아키텍처를 현재 상황에 적응할 수 있도록 재구성하고 컴포넌트 사이의 통신이 가능하도록 커넥터를 설치한다. 재구성이 끝나면 Reconfigurator는 Architecture Broker에게 재구성이 끝났음을 보고하고 Architecture Broker는 Monitor에게 한번의 적응 과정이 완료되었음을 알린다. 적응후 일정 시간이 흐른 후 Evaluator는 적응 행위의 결과를 평가하고 그것을 Learner에게 전달한다.

Learner는 평가 결과를 축적하여 나중에 Decision Maker가 사용할 수 있도록 한다.

위의 과정이 하나의 적응 과정이며 그림 2에 나타나 있다. 다음 장들부터는 각 모듈의 세부적인 기능을 소개한다.

2.2 Architecture Broker

Architecture Broker는 로봇이 문제 상황을 극복하기 위해 로봇 소프트웨어 아키텍처를 변경할 필요가 있을 때 적당한 아키텍처 재구성 전략을 찾는다. 이러한 문제를 해결하기 위해 로봇은 문제를 해결할 수 있는 기능을 제공하는 컴포넌트를 추가하고 교체하거나 제거 함으로서 새로운 아키텍처를 구성할 필요가 있다. 로봇 소프트웨어 아키텍처 재구성은 그림 2와 같이 두 단계로 구성된다. Architecture Broker는 추상적 레벨의 소프트웨어 재구성을 담당하고 Reconfigurator는 선택된 컴포넌트를 이용하여 실제 로봇 소프트웨어 아키텍처를 재구성한다. 이와 같이 컴포넌트의 재구성을 두 레벨로 분리한 것은 첨가 또는 교체되는데 적합한 가능한 많은 후보 컴포넌트들을 검색하여 이용가능하도록 하기 위해서이다.

Architecture Broker는 Monitor에 의해 파악된 상황 온톨로지를 기반으로 문제 해결을 위한 로봇 소프트웨어 재구성 전략을 선택하게 된다. 재구성 전략은 그림 3과 같이 XML 기반의 재구성 언어로 기술되어 있다. XML기반의 아키텍처 재구성 명세는 로봇 소프트웨어 아키텍처 변경 오퍼레이션, 변경되어야 할 추상적 컴포넌트의 이름 그리고 선택 가능한 실제 컴포넌트가 제공해야 할 기능을 기술한다. 이 재구성 전략을 바탕으로 Architecture Broker는 Component Broker에 필요한 컴포넌트의 검색을 요청한다. 그 후 재구성 명세로부터 얻은 재구성 오퍼레이션과 함께 선택된 컴포넌트 명세를 Reconfigurator에 전달한다.

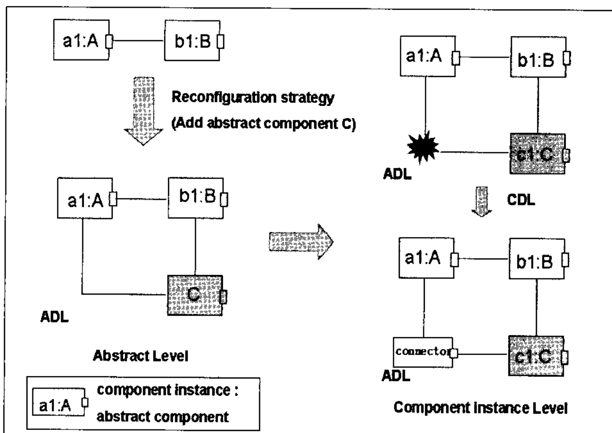


그림 2 두 단계로 구성된 아키텍처 재구성 과정

```
<?xml version="1.0" ?>
<reconfiguarationdescription name="ToVisionbasedLocalization">
   <description>change the current robot architecutre into vision based
   Localizer</description>
   <profile>
         <required slotName="Localizer" action="Replace"/>
         <required slotName="MapBuilder" action="Remove"/>
   </profile>
   <configuration>
         <script>
                  <Replace slotName="Localizer">
                           <services>
                                    <service name="VisionbasedLocalization"/>
                                    <service name='VisionbasedMapBuilding'/>
                           </services>
                  </Replace>
                  <Remove slotName="MapBuilder"/>
         </script>
   </configuration>
</reconfiguarationdescription>
```

그림 3 아키텍처 재구성 전략의 예

2.3 Component Broker

Component Broker는 로봇이 처한 문제 상황을 해결하는데 적합한 컴포넌트를 추론하여 찾아낸다. 우리의 이전의 연구[2, 3]에서는 Component Broker가 상황, 재구성전략, 컴포넌트 온톨로지를 검색하여 컴포넌트의 후보를 추출하였다. 그러나 이번 연구에서는 아키텍처 재구성을 위해 필요한 아키텍처 재구성 명세에 포함되어 있는 컴포넌트들을 검색하는 것으로 확장되었다. Component Broker가 컴포넌트를 검색하는 과정은 다음과 같다. 우선 Architecture Broker가 소프트웨어 아키텍처를 재구성하기 위해 필요한 컴포넌트를 요청하면, Component Broker는 그에 대한 컴포넌트 후보들을 추출하여 Architecture Broker에 전달한다. 만일 컴포넌트 후보들이 내부에 없거나 검색된 후보들 중 적절한 후보가 없다고 판단(이 판단은 Decision Maker에서 한다) 될 시에는 Component Broker는 컴포넌트 획득 엔진에게 로봇 외부의 저장소에서 필요한 컴포넌트들을 검색하고 수집하도록 요청한다.

그림 4는 Architecture Broker와 Component Broker의 동작 과정을 보여준다.

1) Monitor로부터 받은 상황정보를 바탕으로 상황 온톨로지(Situation Ontology)에서 상황 인스턴스가 추론된다.
2) Architecture Broker는 상황 인스턴스에 관계된 아키텍처 재구성 전략의 스키마를 검색한다.
3) Architecture Broker는 선택된 아키텍처 재구성 명세로부터 재구성 오퍼레이션과 필요한 컴포넌트의 기능을 추출한다.
4) Architecture Broker는 아키텍처 재구성 명세에 포함된 컴포넌트의 기능을 파악하고 Component Broker에 요청한다.
5) Component Broker는 기능 온톨로지(Function Ontology)에 명시된 컴포넌트의 기능을 바탕으로 적합한 컴포넌틀를 추론한다.

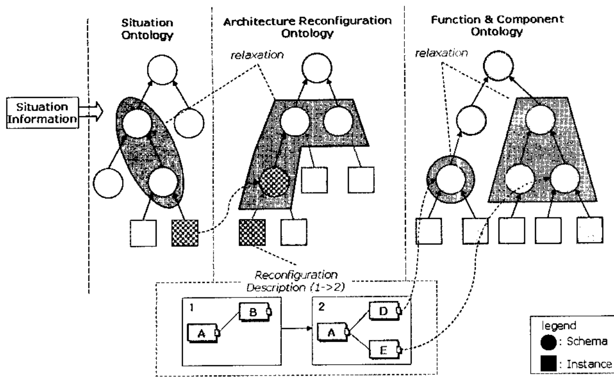


그림 4 온톨로지를 이용한 Architecture Broker 및 Component Broker의 추론 과정

2.4 Decision Maker & Learner

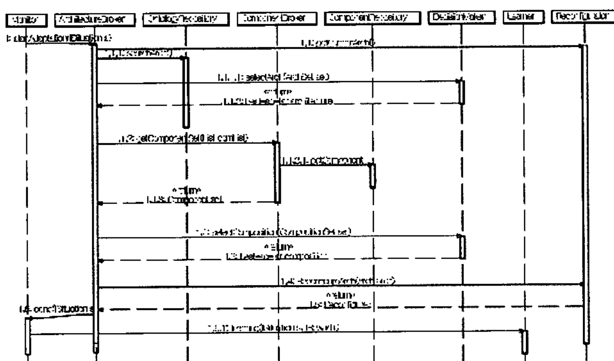


그림 5 SHAGE 프레임워크 내의 각 모듈간의 상호작용

현재 상황에 적합한 software component를 결정하고 학습하기 위하여, Gilboa 와 Schmeidler가 소개한 변형된 CBDT를 사용하였다[4, 5]. CBDT는 유사한 문제들은 유사한 해법을 가진다는 사례기반추론(case-based reasoning)[6]의 아이디어와, learner는 시행착오를 통하여 스스로 개선해 나갈 수 있다는 강화학습(reinforcement learning)[7]의 아이디어에 기반을 둔다. CBDT는 현재 직면한 문제와 유사한 과거의 경험의 수행결과를 기반으로 action(i.e., software component configuration)을 선택한다. 엄밀히 말하면(자세한 설명은 [8]을 참조), P와 A는 각각 problem(robot situation) set과 action set을 나타낸다. problem p에 대하여 action a를 선택하면 $r=r(p,a)\in\Lambda$의 결과가 나타난다. 여기서, Λ는 result set을 의미한다. 이러한 결과로부터 얻는 효용 값은 실수 값을 갖는 효용함수 $u:\Lambda\rightarrow R$를 통하여 주어지고, 여기서, R은 실수집합을 의미한다. problem들 간의 유사도를 측정하기 위하여 다음의 유사도 함수 $\sigma_P : P\times P\rightarrow[0,1]$가 사용되고, result들 간의 유사도 측정은 함수 $\sigma_\Lambda : \Lambda\times\Lambda\rightarrow[0,1]$가 사용된다. 로봇과 관련된 decision maker는, 과거 수행되었던 사례를 저장하는, 제한된 크기의 메모리

$$M=\{(p_1,a_1,r_1),\ldots,(p_n,a_n,r_n)\},$$
$$(p_k,a_k)\in P\times A,\ r_k=(p_k,a_k),\ k=1,\ldots,n$$

를 가진다.

decision maker가 새로운 문제 p'에 직면하였다고 가정하자. 그러면 decision maker는 다음 식을 통하여 지금까지의 경험에 대한 효용 값의 합이 최대가 되게 하는 action을 선택한다.

$$\arg\max_{a\in A} \sum_{(p,a,r)\in M} \sigma_P(p,p')\cdot u(r)$$

간단한 weighted sum에 기반한 이러한 방법은 항상 적절한 action을 선택하지는 못한다. decision maker는 단지 action a가 action b보다 많이 시도되었다는 간단한 이유만으로 b보다 더 좋은 결과를 내지 못하는 a를 선택할 수도 있다. 이러한 현상을 피하기 위하여, 기존의 단순한 weighted sum 대신 "averaged similarity"를 사용한 weighted sum이 사용되었다. 기존의 similarity function σ_P 대신에 다음의 σ^*_P로 교체하여 action에 대한 효용 값의 weighted sum을 계산한다.

$$\sigma^*_P := \sigma_P(p,p')\left(\sum_{(p'',a,r'')\in M}\sigma_P(p'',p)\right)^{-1}$$

하지만, 이러한 방식의 CBDT는 현재 problem과 이전에 경험하였던 사례와의 유사도가 상당히 낮을 경우 적절한 action을 판단하지 못한다. 이러한 문제를 해결하기 위하여 "Satisficing" (satisfice = satisfy + sacrifice) decision making 방식이 [9, 4]에서 소개되었다. 이는 action에 대한 weighted sum의 최대 값이 어떤 주어진 threshold 값보다 클 때까지 exploration을 수행하는 것이다. 하지만 이러한 threshold를 이용한 exploration strategy를 사용하여도, CBDT는 선택한 action이 현재 problem에 대한 optimal action임을 보장하지는 못한다.

CBDT의 sub-optimality 문제를 해결하기 위하여, 본문에서는 강화학습의 ε-greedy exploration-exploitation strategy를 CBDT에 적용하였다. 만약 best action의 weighted sum이 threshold값보다 낮을 경우, 현재 problem 에 대하여 지금까지 적용되지 않았던 action중 하나를 동일한 확률로 랜덤하게 선택한다. 만약 threshold 값보다 크다면, ε의 확률로 지금까지

적용되지 않은 action들 중 하나를 동일한 확률로 랜덤하게 선택하거나 (exploitation), $(1-\varepsilon)$의 확률로 현재까지의 best action을 선택한다(exploration).

2.5 Reconfigurator

Reconfigurator 소프트웨어 아키텍처를 관리하고 실행시간에 중지 없이 재구성한다. 소프트웨어 아키텍처의 재구성은 선택된 아키텍처 재구성 전략과 컴포넌트 구성을 기반으로 수행된다. 실행시간에 재구성을 가능케하기 위해서 SHAGE 프레임워크에서는 소프트웨어 아키텍처를 설계하는 규칙을 정하였다.

SHAGE 프레임워크에서는 소프트웨어 아키텍처는 추상 레벨(abstract level)과 구체화 레벨(concrete level) 두 단계로 정의된다. 추상 레벨에서는 아키텍처에 슬롯(slot)만을 정의하는데 각 슬롯은 서비스를 묘사하는 추상화된 컴포넌트를 표현한다. 각 서비스는 각 슬롯이 제공되어야 하는 추상적 서비스를 정의한다. 이 서비스는 구체적인 코드 수준의 메소드를 지정하는 것은 아니라, 슬롯이 받거나 보낼 수 있는 메시지의 유형을 정의한다. 그림 3에서 나타나 있듯이 각 아키텍처 재구성 전략은 오직 추상 레벨의 아키텍처만을 묘사한다.

```
<?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>
```

그림 6 컴포넌트 기술 언어로 기술한 컴포넌트 기술의 예

구체화 레벨에서는 각 슬롯을 구체화 컴포넌트(concrete component)로 채운다. 각 구체화 컴포넌트는 실행가능한 코드로서 예를 들어 미리 정의된 컴포넌트 구현 규칙에 따라 구현된 자바의 '.class' 파일이 이에 해당된다. 프레임워크에서 사용될 각 컴포넌트는 반드시 'start', 'stop', 'suspend' 등의 공통 메시지를 받을 수 있도록 구현되어야 하고 개별 컴포넌트마다 고유하게 주고 받을 수 있는 메시지의 유형을 그림 ?에서 나타난 예제와 같이 컴포넌트 기술 언어(Component Description Language)로 표현해야 한다.

컴포넌트 기술 언어는 XML 기반의 언어로서 각 컴포넌트가 필요로 하는 인터페이스(required interfaces)와 제공할 수 있는 인터페이스(provided interfaces)를 기술한다. 각 인터페이스는 주거나 받을 수 있는 구체적인 메시지의 유형을 묘사하고, 이 메시지들은 실제로 컴포넌트간의 통신에 사용된다.

아키텍처를 재구성할 때 Reconfigurator는 컴포넌트 사이의 통신이 가능한지 검사한다. 이 때 컴포넌트간의 불일치(mismatches)가 발견되면, 적절한 커넥터를 구성하여 컴포넌트 사이의 불일치를 해소한다. 예를 들어, 두 개의 컴포넌트가 다른 보드에서 실행되어야 한다면 투 컴포넌트 사이에 RMI를 구현하는 커넥터를 설치하여 통신이 가능하도록 한다. 재구성 과정이 완료되면 Reconfigurator는 Architecture Broker에게 재구성이 완료 되었음을 알린다.

2.6 Repositories

온톨로지 저장소, 컴포넌트 저장소, 컴포넌트 획득 엔진, 컴포넌트 퇴거계획이 정의되고 개발되었다[10, 11]. 온톨로지 저장장소는 아키텍처 및 Component Broker에서 필요한 상황, 아키텍처 재구성전략 및 컴포넌트 온톨로지들을 저장하기 위한 공간이다. 컴포넌트 획득 엔진은 로봇의 내부 보유 기능만으로는 문제 상황을 해결할 수 없는 경우에 외부 저장소들로부터 필요한 컴포넌트를 검색하여 수집해 오는 역할을 한다. 그리고 컴포넌트 퇴거 계획은 로봇 내부 저장소가 용량의 한계에 다다랐을 때, 내부의 어떤 컴포넌트 들을 퇴거 시키고 새로운 컴포넌트를 저장할 것인가를 결정하기 위한 계획을 말한다.

그림 7은 저장소의 아키텍처를 보여준다. 외부 획득 엔진은 필요한 아키텍처 재구성 명세나 컴포넌트들이 로봇 내부에 없을 경우 Architecture Broker나 Component Broker로부터 외부 회득 요청을 받는다.

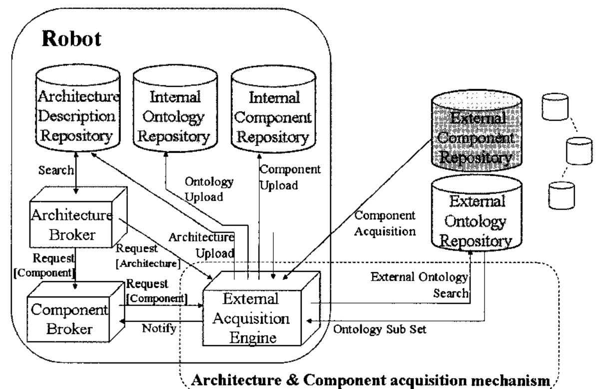


그림 7 프레임워크에서 내/외부 저장소를 이용하는 방법

이러한 요청을 받으면 획득 엔진은 외부 저장소들에서 필요한 정보들을 검색하고 수집하여 내부 저장소에 저장해 주는 역할을 한다.

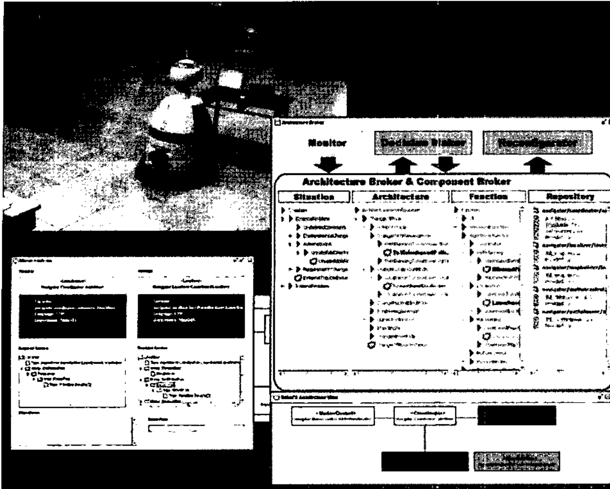
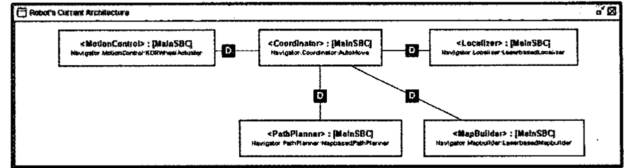


그림 8 실제 로봇을 이용한 실험 장면

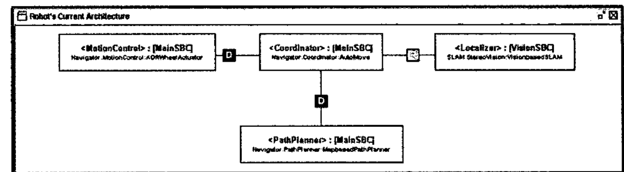
3. 사례 연구

본 연구에서는 프레임워크의 효용성을 검증하기 위해서 실제 로봇에 적용하는 실행을 수행하였다. 실험에 사용된 로봇은 'Infortainment Robot'로 명명된 서비스 로봇의 프로토 타입이다. 이 로봇에는 두 개의 SBC(Single Board Computer)가 설치되어 있다. 하나는 'Main SBC'로서 로봇의 구동부와 레이저, 적외선, 초음파 등의 센서를 관할한다. 다른 하나는 'Vision SBC'로서 카메라로부터 들어온 정보를 처리하도록 설계되어 있다. 두 개의 SBC는 100Mbps 네트워크로 연결되어 있다.

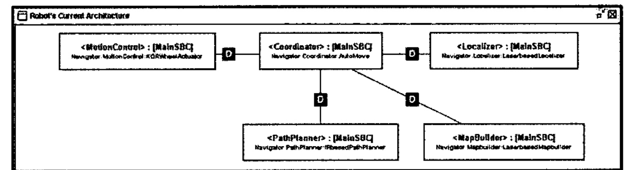
본 사례 연구의 시나리오는 다음과 같다: 1. 초기에 로봇이 '좀더 빨리 주행해야 하는 상황'에 적응된 상태로 시작한다. 이를 위해 로봇은 빠른 인식이 가능한 센서(예를 들어 레이저 센서) 정보를 이용하도록 아키텍처가 구성된다. 2. 이동 중 로봇이 테이블과 같이 아래쪽이 비어있어 레이저 센서 높이로는 인식할 수 없는 장애물에 부딪혀 움직일 수 없는 상황에 직면한다. 그래서 사용자가 로봇에게 좀 더 조심히 움직이라는 요청을 한다. 3. 로봇은 현재의 센서가 인식할 수 없는 물체를 인식할 수 있도록 아키텍처를 재구성할 것을 시도한다. 이 실험을 수행하기 위해 주행에 필요한 컴포넌트들을 구현하였고 그림 ?.(a)와 같이 초기 아키텍처를 구성하였다. 'MotionControl' 컴포넌트는 구동부를 제어하고 'Localizer' 컴포넌트는 로봇의 현재 위치를 파악한다. 'MapBuilder'는 레이저 센서를 기반으로 현재 로봇 주위의 맵을 생성한다. 'PathPlanner'는 현재 위치로부터 목표지점까지의 경로를 생성한다. 'Coordinator'는 다른 컴포넌트의 정보를 전달하고 사용자로부터 목표 지점을 입력받는다.



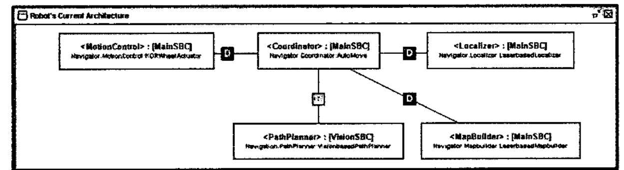
(a)



(b)



(c)



(d)

그림 10 적응 과정에서의 다양한 아키텍처의 모습

초기 아키텍처를 기초로 로봇은 테이블과 같이 아래쪽이 비어있는 장애물을 제외하고는 충돌없이 주행할 수 있다. 실험을 위해서 실험 공간을 설계하였다. 이 공간에는 두 종류의 테이블이 배치되었다. 하나는 테이블보로 덮혀 있어 레이저 센서로도 인식이 가능한 테이블이고 다른 하나는 그림 ?에 보이는바와 같이 보통 테이블로서 레이저 센서로 인식이 불가능한 테이블이다. 이 두 테이블과 로봇을 일직선에 배치하고 로봇에게 두 테이블 사이의 좌표로 목표지점을 주었다. 첫 번째 테이블은 테이블보 때문에 인식이 가능하여 문제없이 회피하여 주행하였다. 그리고 두 번째 목표지점을 테이블보가 없는 테이블 뒤의 좌표로 주었다. 이 때는 로봇이 테이블을 인식하지 못하고 부딪혔다. 그래서 유저는 좀 더 조심해서 주행할 것을 요청하였다. 로봇을 상황이 변화하였으므로 적응을 시도하였다. Monitor가 현재 환경의 상황과 사용자의 요구사항을 Architecture Broker에게 전달하여 적응을 시작하였다. 그림 9.(b)와 (c)에서 나타난 아키텍처 형태로 재구성을 시도하였지만, 여전히 테이블에 부딪히는 문제를 나타내었고,

최종적으로 9.(d)에 나타난 대로 비전 정보 기반의 Path Planner를 사용하는 아키텍처로 재구성하여 테이블을 성공적으로 회피하여 주행하였다. Learner는 상황과 성공적으로 수행했던 경험을 학습하였고, 연속적으로 같은 실험을 수행하였을 때, (a)에서 바로 (d) 아키텍처로 재구성되는 것을 확인할 수 있었다.

4. 요 약

본 논문에서는 내장형 소프트웨어 시스템, 특히 로봇 소프트웨어를 위한 self-managed 소프트웨어 개발 프레임워크로서 SHAGE 프레임워크를 제안하였다. SHAGE 프레임워크는 소프트웨어를 실행시간에 동적으로 변경시킬 수 있도록 지원하는 여러 모듈로 구성되어 있다. Observer가 외부 상황을 관찰하고 관찰된 상황이 Architecture Broker로 전달되면 후보 아키텍처 재구성 전략을 검색하고 Component Broker가 구체화 컴포넌트들을 검색한 후 상황에 적절한 아키텍처 재구성 전략과 컴포넌트 구성을 Decision Maker가 선택한다. Reconfigurator가 선택된 전략과 컴포넌트 구성을 기초로 로봇의 아키텍처를 재구성한다. 적응행위를 Evaluator가 평가하고 그 결과를 Learner가 축적하여 나중에 Decision Maker가 사용할 수 있게 한다.

프레임워크의 효용성을 확인하기 위해서 실제 로봇을 이용한 사례연구를 수행하였고, 이 실험을 통해 적응 과정을 확인하였다. 로봇은 상황과 사용자의 피드백에 적응하였다.

Acknowledgement

본 연구에 도움을 주신 성균관대학교의 이석한 교수님께 감사의 말씀을 드립니다.

참고문헌

[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, "Asemantically-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 Articmlaial 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] E. Hullermeier, "Experience-based decision making: a satisfying decision tree approach," IEEE Transaction on Systems, Man, and Cybernetics, Vol.35, pp.641-653, 2005.

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

[10] 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.

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

박 수 용

1986 서강대학교 전자계산학과(공학사)
1988 플로리다 주립대 전산학(석사)
1995 George Mason University 정보기술학박사, 연구 조교수
1996~1998 TRW Senior Software Engineer
1998~2002 서강대학교 컴퓨터학과 조교수
2002~현재 서강대학교 컴퓨터학과 부교수

관심분야 : 요구공학, Adaptable Components, Web Services

장 형 수

1994 미국 Purdue University, 전기 및 컴퓨터공학과(학사)
1996 미국 Purdue University, 전기 및 컴퓨터공학과(석사)
2001 미국 Purdue University, 전기 및 컴퓨터공학과(박사)
2001. 9~2003. 6 Institute of Systems Reasearch, University of Maryland, College Park, Postdoc.
2002. 6~2003 2 고려대학교 정보통신 기술 공동연구소 연구교수
2003. 3~현재 서강대학교 공과대학 컴퓨터학과 조교수
관심분야 : (Partially Observable) Markov Decision Process, Game Theory with emphasis on Markov Games, Stochastic Modeling and Decision Theory, Computational Learning Theory, Computational Intelligence, Applications of theories in the above areas into Stochastic Resource Control Problems

김 동 선

2003 서강대학교 컴퓨터학(공학사)
2005 서강대학교 컴퓨터학(공학석사)
2005~현재 서강대학교 컴퓨터학과 박사과정
관심분야 : Formal Methods, Embedded Software, Adaptive Software

고 인 영

1990 서강대학교 전산학과(학사)
1992 서강대학교 전산학과(석사)
1993. 6~1994. 5 Lecturer, Korea Air Force Academy
1994. 6~1996. 6 Full-time Lecturer, Korea Air Force Academy
2003 University of Southern California (박사)
2004. 4~2004. 12 Post-doctoral Research Associate, USC Information Sciences Institute (ISI), U.S.A.
2004. 8~2004. 12 Visiting Faculty, Institute for Software Research International (ISRI), School of Computer Science, Carnegie Mellon University, U.S.A.
2004~현재 한국정보통신대학교(ICU) 공학부 조교수
관심분야 : software component management, coordination mechanisms in large-scale, distributed system environments

박 연 출

1999 숭실대학교 컴퓨터학과(석사)
2004 숭실대학교 컴퓨터학과(박사)
2000 (주)애니큐브 연구개발팀 팀장
2003 (주)엠아이엔디 선임연구원
2004 포도시스템즈 CTO
2005 성균관대학교 ISRC Post-Doc
관심분야 : 로봇비전, 이미지프로세싱, 로봇 자가치유, 로봇인지엔진

이 관 우

1994 포항공과대학교 컴퓨터공학과(학사)
1996 포항공과대학교 컴퓨터공학과 (공학석사)
2003 포항공과대학교 컴퓨터공학과 (공학박사)
2003. 9~현재 한성대학교 정보시스템 공학과 조교수
관심분야 : Product Line Engineering, Aspect-Oriented Programming, Dynamic Software Architecture, Real-time Systems Development