제품라인 자산 개발을 위한 휘처 기반 레가시 시스템 재공학 : 홈서비스 로봇 사례연구†

포항공과대학교 이재준 · 김병길 · 장 엽 · 강교철

1. 서 론

홈서비스 로봇은 서비스를 제공하기 위해서 음성 인식이나 영상 처리와 같은 기술 집약적인 컴포넌트를 이용한다. 하지만 홈서비스 로봇 시장이 완전히 정착된 것이 아니기 때문에 변화의 여지가 많다. 시장의 요구가 달라짐에 따라, 기술적인 컴포넌트들은 빈번한 변화를 겪고 있고, 새로운 서비스들이 추가되며, 기존의 서비스들은 제거되거나 변경되고 있다. 이런 변화가 심한 시장에서 효율적으로 경쟁하기 위해, 개발자들은 적은 노력으로 제품을 빠르게 향상시킬 수 있어야 한다. 이를 위해 홈서비스 로봇 산업은 어플리케이션이 쉽게 향상될 수 있는 소프트웨어 개발 프레임워크를 가져야 한다. 따라서 소프트웨어 제품라인 프레임워크는 홈서비스 로봇 산업에 이상적인 개발 프레임워크이다.

제한된 개발 자원 때문에 로봇 개발자들은 기술 집약적인 컴포넌트 개발에 초점을 맞추고, 변화하는 요구사항에 효과적으로 대처할 수 있는 방법은 고려하지 못했다. 아키텍처에 대한 고려가 부족했기 때문에 개발된 기술적인 컴포넌트들은 ad-hoc한 방법에 의해 통합되었고, 만들어진 제품은 휘처상호작용 문제로 인하여 오동작할 뿐만 아니라, 유지/보수가 어려웠다. 레가시 로봇 어플리케이션을 제품라인의 자산으로 재공학하는 것은 개발 비용을 낮출 수 있고, 시스템의 유연성을 높여 로봇 제품의 경쟁력을 높일 수 있다. Jean-Marc 등은 [1, 2, 6]에서 초기 제품라인 자산의 복원을 위해 아키텍처 중심의 재공학을 하라고 제안했다. 이러한 접근은 소프트웨어 아키텍처가 영역의 개념과 관계를 이해하기 위한 중요한 요소라고 강조하고 있다. Bosch 등은 [3, 4]에서 휘처 모델을 레가시 제품에서 제품라인 자산을 만들어내는 핵심 요소로 강조했다. 하지만 이러한 연구들은 제품라인 자산이 적응성을 갖게 하기 위한 구체적인 디자인 원리나 가이드라인을 제시하지 못하고 있다.

이 논문에서는, 공학적 원칙과 가이드라인[5]에 의한 휘처 기반 방법론을 통해서, 홈서비스 로봇 어플리케이션을 제품라인 자산으로 재공학한 경험에 대해서 설명하고자한다. 먼저, 레가시 로봇 어플리케이션에서 컴포넌트와 아키텍처 정보를 추출[7]하였다. 그 다음 추출한 정보와 도메인 정보를 바탕으로 로봇 어플리케이션의 휘처들을 찾아내었다. 시장의 요구와 기술 변화에 의한 어플리케이션의 변화를 예측하여, 추가적인 휘처와 가변성[8]을 표시함으로써 휘처 모델을 정제하였다. 마지막으로 정제된 휘처 모델과 세 가지의 엔지니어링 원리를 기반으로 하여 진화에 적합한 제품라인[9]을 개발하였다. 이를 위해 새로운 아키텍처와 컴포넌트들을 재설계하였다. 그림 1은 이러한 재공학적 접근 방법에 대해 설명하고 있다.

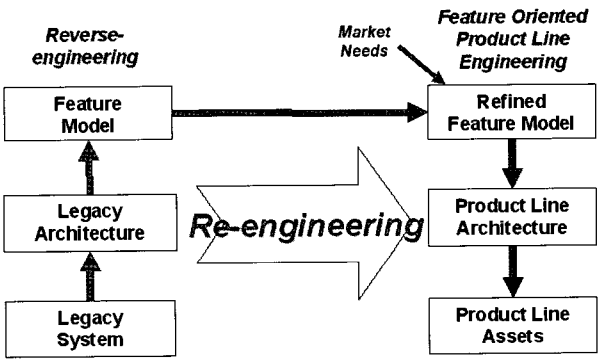


그림 1 재공학 과정의 개관

앞으로의 논문의 내용은 다음과 같다. 우선 2장에서 홈서비스 로봇에 대해 설명한다. 3장에서 레가시 어플리케이션으로부터 아키텍처 정보를 추출하는 과정에 대해서 설명하고, 4장에서 레가시 어플리케이션으로부터 휘처 모델을 추출하고 정제하는 과정에 대해서 설명한다. 5장에서는 정제된 휘처 모델을 기반으로 아키텍처와 자산 컴포넌트들을 엔지니어링 원리에 입각하

† 본 논문은 9th International Software Product Line Conference (SPLC Europe) Rennes, France, September 26-29, 2005, pp.45-56. (LNCS 3714) 에 게재된 논문을 번역, 일부 수정한 것입니다.

여 재설계하는 것을 보여준다. 6장에서는 재공학으로 얻어진 제품라인 자산을 검증한다. 7장에서는 이번 프로젝트를 수행하면서 얻은 교훈에 대해서 설명한다. 마지막 8장에서는 논문을 요약하고, 향후 연구에 대해 언급한다.

2. 홈서비스 로봇의 배경

이 장에서는 제품라인 자산으로 재공학한 홈서비스 로봇(HSR)에 대해서 알아보도록 하겠다. HSR은 주택 감시, 청소 등 일상적인 집안일을 돕기 위해 개발되었다. 우리는 HSR 개발자로부터 HSR 서비스에 대한 high-level 명세를 건네받았다. 예를 들어 "call and come"(사용자의 위치를 파악하고 다가옴)이나 "user following"(사용자를 계속 따라다님), ""security monitoring" (주택 감시), "tele-presence"(HSR을 원격으로 조작) 등이 있다. 이 외에도 2개의 분리된 HSR 어플리케이션을 받았는데, 이는 각각 "call and come"이나 "user following"을 구현하고 있다. 이런 기본적인 서비스들 중에서 "call and come"과 "user following"에 대해서 자세하게 알아보겠다.

- **Call and Come(CC)**

이 서비스는 부착된 마이크로폰으로 입력 받은 소리를 분석하고, 미리 정의된 패턴과 일치하는지 확인한다(예: 박수소리나 음성 명령). 정의된 패턴으로 "stop"과 "come"의 2개 명령이 있다. "come" 명령을 인식하면, 로봇은 소리의 근원을 찾는다. 그 후 탐지한 소리의 방향으로 회전한 다음, 전향 카메라에 의해 촬영된 영상 데이터를 분석하여 사람의 얼굴을 인지한다. 사용자의 얼굴을 찾으면, 로봇은 사람에게 1미터 이내로 접근할 때까지 이동한다(사용자와의 거리는 구조화된 빛 센서로 측정한다.). "stop" 명령은 로봇을 멈추게 한다. 명령 인식, 소리 근원 찾기, 사용자 얼굴 인식 등에 실패하면, CC는 초기 상태로 되며, 새로운 명령을 기다리게 된다.

- **User Following(UF)**

로봇은 사용자의 위치를 파악하기 위해서 전향 카메라와 구조화된 빛 센서를 이용한다. UF가 시작하면, 로봇은 사용자의 위치를 판단하기 위해서 구조화된 빛 센서로부터 들어온 데이터와 영상 데이터를 계속적으로 체크한다. 로봇은 1미터 간격으로 사용자를 계속 따라간다. 만약 사용자를 놓치게 된다면, 로봇은 UF가 종료되었다는 음성 메시지를 출력한다. 사용자는 "come" 명령을 이용하여 로봇이 사용자를 인식하고 UF를 재개할 수 있도록 할 수 있다.

주어진 명세와 2개의 레가시 어플리케이션에 기반하여, 우리는 어플리케이션들을 커버할 수 있는 후보 휘처들을 추출했다. 레가시 HSR 어플리케이션은 미래에 있을 수 있는 확장이나 변화에 용이하지 않게 구현되어 있었다. 예를 들어 레가시 HSR 어플리케이션은 사용자를 찾고 뒤쫓아가기 위해 "Face Detection Method"와 "Object Recognition With SL" 휘처를 가지고 있다. 하지만 이런 휘처들은 가변성을 제대로 반영하지 못하고 있었다. 예를 들어, "Face Detection Method"는 "Color-based" 방법을 이용하고 있는데, 이는 다른 탐지기술로 대치할 수 없게 되어있다. 레가시 HSR 어플리케이션의 휘처에 대한 자세한 설명은 그림 5에 설명되어있다.

3. 레가시 HSR 어플리케이션으로부터 정보 추출

이번 장에서는, 레가시 어플리케이션으로부터 아키텍처 정보를 추출하는 방법과, 추출된 아키텍처에 잠재된 문제가 무엇인지에대해서 설명한다.

3.1 역공학 과정

레가시 어플리케이션으로부터 개념적 아키텍처와 프로세스 아키텍처를 추출한 과정을 그림 2에서 설명하고 있다.

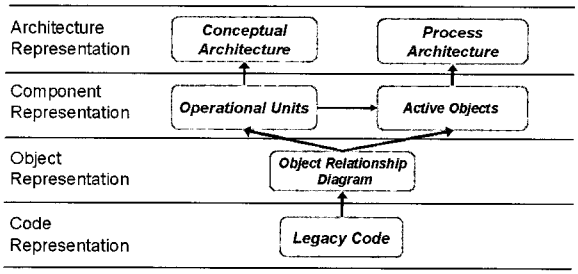


그림 2 개념적 아키텍처와 프로세스 아키텍처 복원 과정

1) 레가시 어플리케이션으로부터 Rational-Rose tool을 이용하여 객체 관계 다이어그램을 추출했다.
2) 추출된 객체 관계 다이어그램을 기반으로, 서비스를 구성하는 객체들을 결정했다(예: CC 와 UF 서비스). 이 과정에서 도메인 지식이나 데이터 흐름 분석을 기반으로 하는 경험적인 방법(heuristic)을 사용하였다. 그 후에, 메소드 호출이나 데이터 흐름을 분석하여 서비스를 구성하는 수행 단위를 찾아내었다. 이렇게 찾은 수행 단위를 아키텍처 컴포넌트로 대응시켜 개념적 아키텍처를 복원하였다.
3) 객체 관계 다이어그램과 찾아낸 서비스/수행 단위로부터, 프로세스/스레드를 만들어 다른 객체의

수행을 지시하는 객체(활성 객체)를 찾았다. 그리고 제어 흐름을 분석함으로써, 활성 객체 사이의 상호작용을 확인했다. 이렇게 활성 객체 사이의 상호작용을 파악함으로 소프트웨어 컴포넌트의 할당을 보여주는 프로세서 아키텍처를 복원하였다.

3.2 실행 단위의 추출

그림 3은 CC의 객체 관계 다이어그램에서 수행 단위를 추출하는 과정을 보여준다. 기능적 응집도를 기준으로, 실행 단위를 3 종류의 카테고리로 나누었다. sensor(입력), controller (조정자), actuator(출력). 이러한 카테고리를 이용하여 다음과 같이 5개의 실행 단위를 구별하였다.

- sensor units: "Face Detection", "SL Sensing", "Clap Recognition",
- a controller unit: "CC Command Controller".
- an actuator unit: "Actuator Controller"

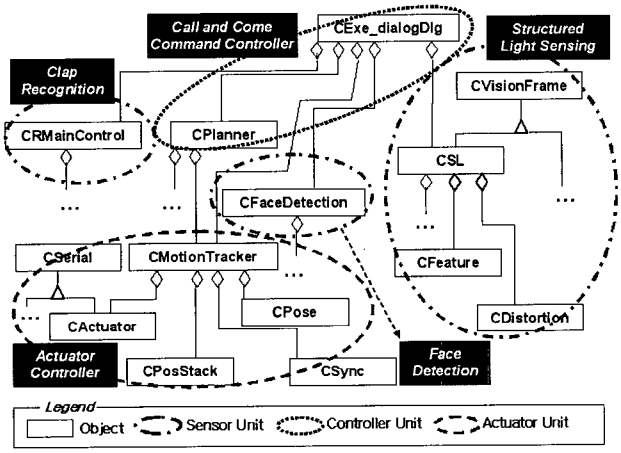


그림 3 CC의 수행단위 추출

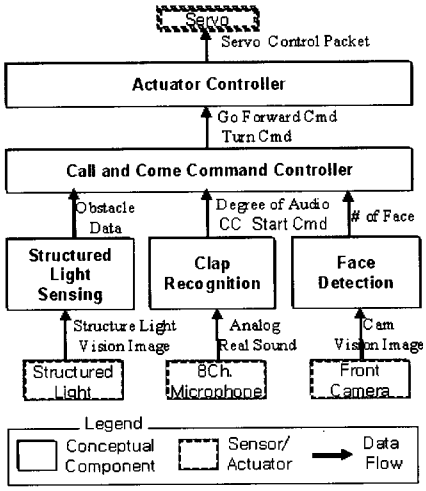
3.3 개념적 아키텍처와 프로세스 아키텍처의 복원

그림 4 (a)에서 알 수 있듯이, 추가적인 데이터 흐름 분석을 통해서 실행 단위들이 그림 4 (a)의 개념적 아키텍처에 할당되었다. 이 아키텍처는 멀티 서비스 로봇에 적합하지 않았는데, 모든 서비스 단위들이(예: CC 명령 컨트롤러) "Actuator Controller"를 직접적으로 접근 가능했고, 제어 또한 가능했기 때문이었다. 때문에 이러한 아키텍처에서는 서비스들이 서로의 동작을 방해 할 수 있는 가능성이 있었다.

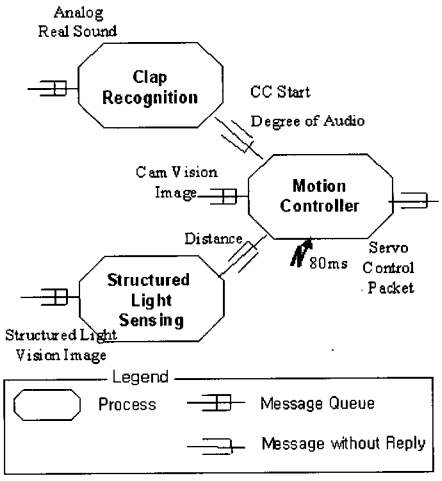
우리는 프로세스 아키텍처를 복원하기 위해 CEXE_dialog, CRMainControl, CSL의 세 프로세스 생성 코드를 분석하여 세 활성 객체들을 찾았다. 이 활성 객체들은 "Motion Controller(MC)", "Clap Recognition(CR)", "SL Sensing(SLS)"의 세 프로세스를 생성한다. MC는 "CC Command Controller"과 "Face Detection", "Actuator Controller" 수행 단위들로 구성되어 있고, CR은 "Clap Recognition" 단위, SLS는 "Structured Light Sensing" 단위로 구성되어 있다. MC는 장애물까지의 거리와 박수 소리의 방향에 관한 데이터를 각각 SLS와 CR로부터 받는다. MC는 이런 데이터를 처리하여 움직여야 할 방향을 결정한다. 그러므로 MC에 적절한 컨트롤 로직이 없다면, CR 과 SLS사이의 휘처상호작용이 일어날 것이다. 두 개의 프로세서가 MC를 동시에 제어할 수 있기 때문이다.

4. HSR 제품라인의 정련된 휘처 모델

이 장에는 정련된 HSR의 휘처 모델에 대해 설명하도록 하겠다. 우선 CC 서비스를 구현한 부분에서 휘처들을 추출하였다. 이 휘처들은 그림 5에서 BOLD체



a) 개념적 아키텍처



b) 프로세스 아키텍처

그림 4 복원된 개념적 아키텍처와 프로세스 아키텍처

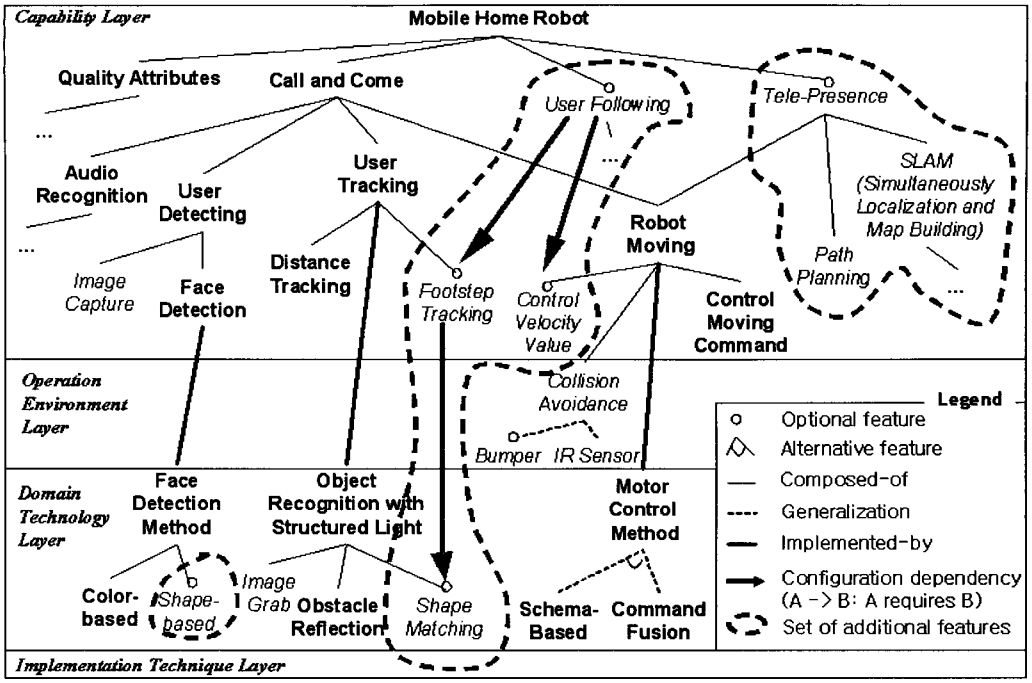


그림 5 CC 서비스를 포함하는 HSR의 휘처모델

로 쓰여 있다. 새롭게 추가되거나 정련된 휘처들은 이탤릭체로 쓰여 있다. 정련된 휘처 모델에 대한 자세한 설명은 다음과 같다.

우선 우리는 새로운 시장을 목표로 하여 새로운 서비스를 추가하였다. CC 서비스만을 수행할 수 있어 물건을 옮기는 기능 등을 담당하는 HSR의 경우 lowend 시장이 대상이지만, CC, UF, tele-presence, security monitoring 서비스를 수행할 수 있는 HSR은 지능형 도우미와 같은 high-end 시장을 목표로 할 수 있다. CC 서비스만을 지원하는 기존의 프로그램을 바탕으로 만든 휘처 모델로부터 우리는 새로운 서비스, 오퍼레이션, 전문 기술과 관련된 휘처들과 휘처들의 의존성 관계를 추가하였다. 새롭게 추가된 서비스들은 기존의 휘처 모델에 포함되지 않았던 오퍼레이션 휘처들을 요구한다. 예를 들면 새롭게 추가된 UF 서비스를 수행하기 위하여 사용자의 움직임을 추적할 수 있어야 하고("Footstep Tracking"), 사용자를 자연스럽게 따라다닐 수 있기 위하여 HSR의 속도를 조절할 수 있어야 한다("Control Velocity Value"). 또한 새로운 오퍼레이션휘처는 새로운 전문 영역 기술을 요구로 하는데 예를 들어 "Footstep Tracking"은 사용자의 움직임을 인식하기 위하여 "Shape matching"을 요구한다.

그 다음 우리는 예측되는 변화를 수용하기 위하여 선택적 휘처를 추가하는 방식으로 휘처 모델을 정련하였다. 기존 프로그램에서 CC 서비스를 수행하기 위하여 이용되는 "Face Detection Method"는 단지 색을 기반하여 사용자의 얼굴을 찾고 있었다. 우리는 자원의 소모가 커지더라도 사용자 인식의 정확성을 높이기를 원하는 시장의 요구를 수용하기 위하여 "Shape-based" 휘처를 추가하였다.

마지막으로 우리는 기술의 발전을 고려하여 휘처 모델을 정련하였다. 능력 계층에 존재하는 휘처 중 일부는 기술의 발전에 의해 OS나 SoC(System on Chip)와 같은 운영환경에 의해 그 기능이 제공될 수 있다. 예를 들어 "Collision Avoidance" 휘처는 능력계층에 존재하는 휘처로 소프트웨어가 구현하는 기능이었으나, 기술의 발전으로 "Collision Avoidance" 기능을 시중의 SoC가 제공할 수 있게 되었기 때문에 운영환경 계층으로 옮겼다.

5. HSR의 새 아키텍처 설계

우리는 HSR의 새 아키텍처를 기술의 발달이나 서비스 명세의 변화로 인한 컴포넌트들의 교체에 용이하도록 유연성을 중시하여 설계하였다. 이를 위하여 새 아키텍처는 컴포넌트의 교체에 용이한 C2 아키텍처 스타일[10]을 확장하여 설계되었고, 컴포넌트와 휘처들은 휘처의 선택/삭제에 따른 컴포넌트의 추가/변경/삭제에 용이하도록 1:N 관계로 맵핑하였다. 또한 기존 어플리케이션과 정련된 휘처 모델(그림 5)을 분석한 결과를 바탕으로 세 가지 공학적 원칙[9]을 적용하여 HSR의 새 아키텍처를 설계하였다.

우선, 기존의 아키텍처는 제어 컴포넌트와 연산 컴

포넌트의 기능이 혼재되어 있어, 어플리케이션의 행위를 분석하는데 어려움을 주고 있었다. 따라서 우리는 첫 번째 원칙인 "연산 관점과 제어 관점을 분리"를 적용하여 이 문제를 해결하려 하였다. 연산 컴포넌트들로 이루어진 연산 영역과, 제어 컴포넌트들로 이루어진 제어 영역을 구별함으로써 데이터의 흐름과 제어의 흐름을 뚜렷이 구별할 수 있게 되었고, 이는 시스템의 행위를 분석하는데 있어 큰 도움을 주게 되었다. 또한 컴포넌트가 담당하는 역할이 명백해짐으로써 컴포넌트의 추가/삭제가 편하게 되었다.

둘째, 우리는 서비스의 추가/삭제가 다른 서비스에 미칠 수 있는 여파를 최소화하려고 노력하였다. 휘처들 간의 관계에 대한 고려 없이 컴포넌트들을 통합하여 서비스를 추가하는 것은 휘처상호작용 문제를 유발하게 된다. 기존의 아키텍처는 서비스 컴포넌트들 간의 상호 관계에 대해 세심하게 배려하지 않았기 때문에 새로운 서비스를 추가할 때마다 휘처상호작용 문제가 발생하였다. 이 문제를 해결하기 위하여 두 번째 공학적 원칙인 "전역 행위와 지역 행위의 분리"를 적용하였다. 이를 적용하여 서비스 컴포넌트들은 다른 서비스 컴포넌트와 독립적으로 수행될 수 있도록 따로 분리되었다. 이로써 컴포넌트의 추가/삭제가 다른 컴포넌트들에 미치는 영향을 최소화할 수 있게 되어 변화에 용이하게 되었다. 이러한 컴포넌트들을 중재할 수 있도록 Mode Manager를 설계하여 시스템의 전역 행위를 통제하여 서비스들 간의 휘처상호작용 문제를 해결하였다.

마지막으로 우리는 일부 휘처들 사이에 계층이 존재하는 것을 발견하였다. 예를 들어 "Object Recognition with SL"휘처는 세 종류의 자식 휘처-"Image Grab"과 "Obstacle Reflection", "Shape Matching"-를 가지고 있다. "Image Grab"은 구조화된 빛 센서로부터 얻어진 영상을 그대로 사용하며, "Obstacle Reflection"은 "Image Grab"으로 얻어진 데이터를 HSR의 앞에 있는 장애물을 파악하기 위하여 분석한다. "Shape Matching"은 사용자의 다리를 인식하여 사용자를 추적하는 기능 등에 이용되는데 "Obstacle Reflection"에서 얻어진 정련된 객체 이미지들을 이용한다. 우리는 세 번째 원칙 "데이터 정련에 따른 계층설계"를 적용하여 휘처들에 대응되도록 컴포넌트 내의 계층을 세 구간으로 나누었다. 각기 다른 서비스들은 컴포넌트 내의 다른 계층의 데이터를 요구할 수 있다. 연산 컴포넌트 내부에 계층적 아키텍처를 도입함으로써 영역 기술 계층의 휘처 변화를 잘 정의된 계층 간의 인터페이스를 통하여 간단히 구현할 수 있었다.

그림 6은 세 가지 공학 원칙을 바탕으로 재설계한 아키텍처를 보여주고 있다. 우선 우리는 CC와 UF, Tele-Presence, Security Monitoring 네 개의 제어 컴포넌트와 Navigation과 Structured Light, User Interface, Vision Manager, Audio Manager 다섯 개의 연산 컴포넌트를 찾았다. 그리고 모든 컴포넌트들로부터 Up-stream 이벤트를 받아 전역 행위를 다루는 Mode Manager 컴포넌트를 명세하였다. 대부분의 연산 컴포넌트들은 센서들로부터 미가공 데이터를 입력받아 다른 컴포넌트들이 이용할 수 있도록 가공하는 역할을 담당하고 있으며 가공된 데이터들은 제어 컴포넌트에게 데이터 커넥터와 버스를 통하여 전달된다.

우리는 매크로 프로세싱을 통하여 가변적인 휘처들을 수용할 수 있도록 컴포넌트를 설계하였으며[12], 리팩토링을 통하여 기존의 코드로부터 자식 컴포넌트들을 추출하였다. 그림 7의 좌측 그림은 계층이 있는 연산 컴포넌트를 위한 템플릿과 템플릿을 이용한 Structured Light 컴포넌트를 보여주고 있다. 기존의

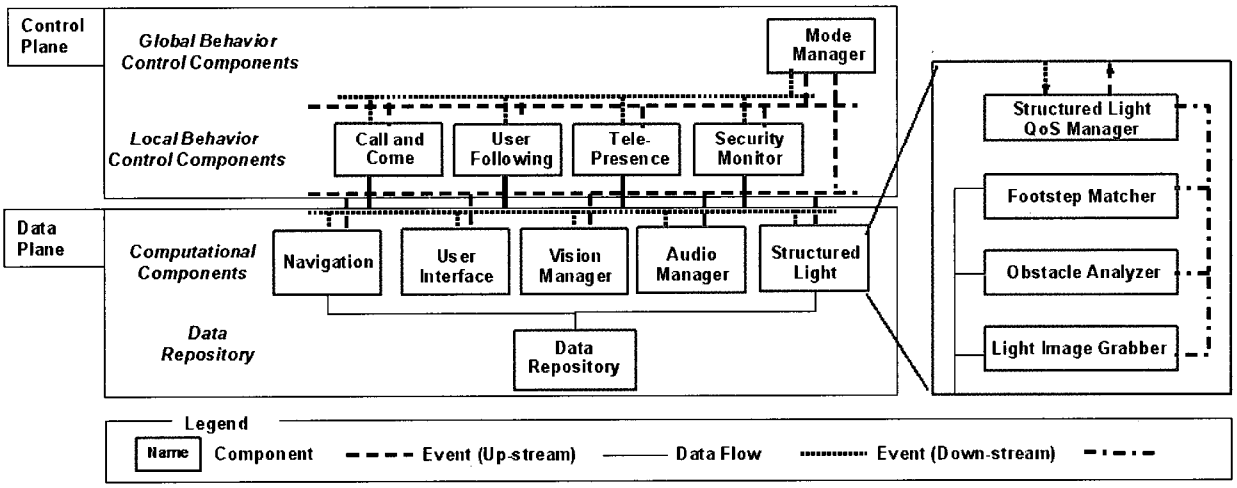


그림 6 HSR을 위한 새 아키텍처

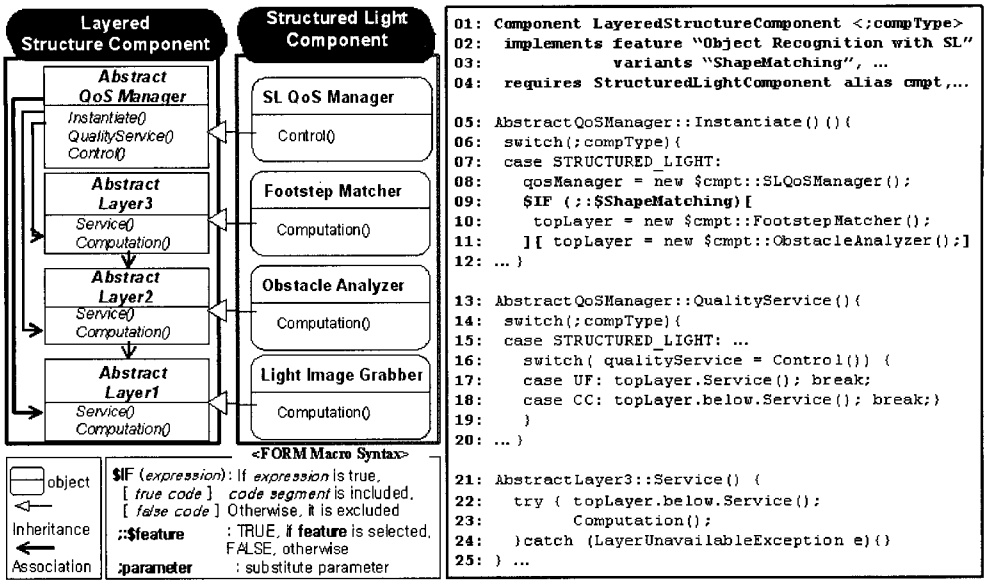


그림 7 설계 객체 모델과 컴포넌트 명세

Structured Light 컴포넌트는 매우 긴 하나의 함수로 이루어져 있었기 때문에 우리는 "Footstep Matcher"와 "Obstacle Analyzer", "Light Image Grabber"같이 재사용이 가능한 부분을 찾아 분리하였다. 이러한 계층화된 컴포넌트는 컴포넌트 명세를 이용하여 선택된 휘처를 바탕으로 생성된다[14].

그림 7의 우측 1~4번째 줄은 LayeredStructure-Component가 휘처 "Object Recognition with SL"과 가변적 휘처인 "Shape Matching"을 구현하고 있음을 의미한다. Line 5~12번째 줄은 Structured Light 컴포넌트가 생성되는 부분을 묘사하고 있는데, 그 중 9번째 줄부터 11번째 줄 부분은 가변적 휘처인 "Shape Matching"이 선택되었을 경우 최상위 계층이 "Footstep Matcher"가 되고, 선택되지 않았을 경우에 최상위 계층이 "Obstacle Analyzer"가 되는 것을 보여주고 있다. 13~20번째 줄은 다른 제어 컴포넌트가 Structured Light 컴포넌트에게 데이터를 요구할 때, 어떤 서비스가 수행되는지 보여주고 있다. 가운데 17번 째 줄은 UF가 Structured Light 컴포넌트에게 서비스를 요구하였을 경우 최상위 계층("Footstep Matcher")의 서비스가 제공되어야 함을 보여주고 있다. 마지막 21~24번째 줄은 계층 간의 서비스 연결을 보여주고 있다.

6. 제품라인 자산의 유효성 평가

우리는 재공학한 제품라인 자산을 바탕으로 HSR 어플리케이션을 생성하였다. 큰 어려움 없이 휘처의 선택으로 CC와 UF 서비스를 수행할 수 있는 두 개의 어플리케이션을 생성할 수 있었으며, 각 서비스들이 명세와 동일하게 수행됨을 확인할 수 있었다. 두 어플리케이션은 하나의 서비스만을 수행하기 때문에 전역 행위를 제어하는 역할을 맡고 있는 Mode Manager의 기능이 특별히 요구되고 있지는 않았다.

그 다음 CC와 UF 서비스를 모두 지원할 수 있는 어플리케이션을 생성하였다. CC와 UF 서비스는 연산 컴포넌트 중 일부를 공유하고 있으며 "Navigation"을 제외하고 휘처상호작용 문제가 발생하지 않았다 CC와 UF는 대부분의 연산 컴포넌트들에게 분석된 데이터 읽기를 요구하였으며 데이터 쓰기를 요구하지는 않았다. 또한 두 서비스는 필요로 하는 데이터의 정련 계층이 달랐는데, 예를 들어 CC는 "Structured Light" 컴포넌트의 "Obstacle analyzer" 계층의 데이터를 요구하는 반면에 UF는 "Footstep Matcher" 계층의 데이터를 필요로 하였다. 이와 달리 CC와 UF가 "Navigation" 컴포넌트에 요구하는 오퍼레이션들은 대부분 작동기를 제어하는 기능이여서 휘처상호작용 문제가 발생하였다. 이를 해결하기 위하여 Mode Manager가 CC와 UF의 우선순위를 조절해야 한다. 우선순위의 변경시 코드의 수정이 CC와 UF에서는 일어나지 않았으며 Mode Manager 에서만 이루어지게 된다. 따라서 재공학을 거친 HSR의 제품라인 자산이 홈서비스 로봇의 어플리케이션을 만드는데 있어 유용하다고 볼 수 있다.

7. 교 훈

이 장에서는 홈서비스 로봇을 재공학하면서 얻을 수

있었던 교훈에 대해서 설명하도록 하겠다.

7.1 계획된 자산 통합의 중요성

하드웨어나 기술 개발을 중심으로 하는 조직에서는 제품 개발을 모든 컴포넌트들을 개발한 이후 단순히 개발된 컴포넌트들을 통합함으로써 이뤄지는 것으로 생각하는 경향이 있다. 하지만 컴포넌트 통합 계획과 아키텍처에 대한 고려가 없다면 개발된 제품은 휘처상호작용 문제를 겪고 제품의 유지 보수비용이 증가하게 된다.

이러한 문제점들을 해결하기 위하여, 우리는 정제된 휘처 모델과 제품라인 자산을 개발하는데 이용될 수 있는 공학적 원칙에 기반을 둔 아키텍처 프레임워크를 제시하였다. 또한 휘처와 아키텍처 컴포넌트 사이에 맵핑 관계를 부여하여 휘처의 선택/제외가 제품에 미치는 영향을 명확하게 하였다. 또한 기존의 휘처들과 새 휘처들의 관계를 파악하는데 휘처 모델이 중요한 역할을 담당하고 있음을 발견하였다. 예를 들어 그림 5의 휘처 모델을 보면, "User Following" 휘처를 추가하기 위해서 필요한 "Footstep Tracking"과 같이 새로운 휘처들뿐만 아니라, 새로운 휘처들과 기존 휘처들의 관계를 파악하는데 큰 도움을 얻을 수 있음을 확인할 수 있다.

휘처 분석 결과를 바탕으로 하여 우리는 컴포넌트 통합에 대한 방법을 결정할 수 있었는데, 예를 들어 "Footstep Tracking" 휘처를 통합할 때, "User Tracking"을 구현한 컴포넌트가 "Footstep Tracking" 휘처에 적합하게 수정되어야 하며, 변화를 수용하기 위하여 "Distance Tracking"과 "Footstep Tracking"에게 공통으로 사용될 수 있는 인터페이스를 제공해야 함을 찾을 수 있었다.

7.2 아키텍처 계층화에서 휘처 모델의 이점

해당 사례 연구를 통하여 우리는 휘처 모델이 컴포넌트 아키텍처의 계층을 파악하는데 유용한 정보를 줄 수 있음을 알 수 있었다. 휘처 모델은 각 서비스가 수행될 때 필요로 하는 다른 수준의 연산 과정을 의미하는 휘처를 가질 수 있다. 예를 들어 "Shape Matching"과 "Obstacle Reflection", "Image Grab" 휘처들은 UF, CC 와 Tele-Presence를 수행하기 위해 사용된다(그림 5 참조). 이 휘처들은 연산 과정의 단계를 나타내는데 "Shape Matching"은 "Obstacle Reflection"연산의 결과를 필요로 하고, "Obstacle Reflection"은 "Image Grab"의 연산 결과를 필요로 한다. 따라서 이 휘처들은 각각 Structured Light Component 내부의 "Footstep Matcher" 계층, "Obstacle Analyzer" 계층, "light Image Grabber" 계층으로 구현되었다(그림 7 참조). 우리는 "Face Detection Method" 휘처도 자식 휘처들과 동일한 관계를 맺고 있음을 확인할 수 있었고, 이 정보를 바탕으로 "Vision Manager" 컴포넌트를 설계할 수 있었다. 따라서 휘처 모델에 근거한 데이터 정련 과정의 계층화는 제품라인 공학에서 컴포넌트 아키텍처를 설계할 때 유용하다는 것을 확인할 수 있었다.

7.3 프로세스 아키텍처가 분석에 주는 이점

우리는 프로세스 아키텍처가 기존 프로그램의 이해에 도움을 줌과 동시에 동시성을 지닌 프로세스 간의 휘처상호작용 문제를 해결하는데 도움을 줄 수 있음을 발견하였다. 예를 들어 기존 프로그램에서 추출한 그림 4 (b)의 프로세스 아키텍처로부터 MC 가 CR과 SLS의 입력 값으로 인한 휘처상호작용 문제를 겪고 있음을 추측할 수 있었다. 또한 우리는 UF 서비스 명세와 다르게 UF 서비스가 전향 카메라를 사용하지 않는 이유를 발견할 수 있었다. UF 서비스가 전향 카메라를 이용하기 위해서는 전향 카메라는 끊임없이 사용자의 얼굴을 촬영해야 한다. 하지만 "Face Detection" 수행단위가 독립적인 수행단위가 아니라 MC 내부에서 순차적으로 동작하는 컴포넌트 중 하나로 구성되어 있었기 때문에 UF 어플리케이션이 전향 카메라를 사용하지 못하였다.

8. 결 론

이 논문에서 우리는 기존의 가정용 서비스 로봇 프로그램을 휘처 기반 방법론을 통하여 제품라인 자산으로 재공학하는 과정에 대해서 살펴보았다. 본 사례 연구를 통하여 휘처 기반 재공학 방법은 로봇 개발자들이 제품라인을 개발하는데 큰 도움을 주며, 개발 단가를 낮추고 프로그램의 유연성을 증대 시키는데 도움을 줄 수 있음을 발견하였다.

향후, 우리는 재공학 과정을 거친 홈서비스 로봇 제품 자산을 바탕으로 HSR 제품라인 자산의 진화에 대해 연구하여 현재 개발된 제품라인 자산의 장단점을 평가할 것이다. 또한 개발된 제품라인 자산을 평가할 수 있는 가이드라인에 대해 조사하도록 할 것이다.

참고문헌

[1] DeBaud, J.M., Girard, J.F.: The relationship between the Product Line Development

Entry Points and Reengineering, 2nd International Workshop on Development and Evolution of Software Architectures for Product Families, LNCS 1492, pp. 132-139, 1998.
[2] Bayer, J., Girard, J.F, Wuerthner, M., DeBaud, J.M., Apel, M.: Transitioning Legacy Assets to a Product Line Architecture, 7th European Software Engineering Conference (ESEC/FSE'99), LNCS-1687, pp.446-463, 1999.
[3] Bosch, J., Ran, A.: Evolution of Software Product Families, Software Architectures for Product Families: International Workshop (IW-SAPF-3), LNCS 1952, pp.168-183, 2000.
[4] Maccari, A., Riva, C.: Architectural Evolution of Legacy Product Families, Software Product Family Engineering: 4th International Workshop (PFE-4 2001), LNCS 2290, pp.64-69, 2002.
[5] Kang, K., Lee, J., Donohoe, P.: Feature Oriented Product Line Engineering, IEEE Software, 19(4), July/August, pp.58-65, 2002.
[6] Eixelsberger, W., Kalan, M., Ogris, M., Beckman, H., Bellay, B., Gall, H.: Recovery of Architectural Structure: A Case Study, 2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families LNCS Vol. 1429. Springer-Verlag, Berlin Heidelberg New York, 2002.
[7] Bergey, J., O'Brien, L., Smith, D.: Option Analysis for Reengineering (OAR): A Method for Mining Legacy Assets (CMU/SEI-2001-TN-013). Pittsburgh, PA:Software Engineering Institute, Carnegie Mellon University, 2001.
[8] Lee, K., Kang, K., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Gacek, C. (eds.): Software Reuse: Methods, Techniques, and Tools.Lecture Notes in Computer Science, Vol. 2319. Springer-Verlag, Berlin Heidelberg, 2002.
[9] Kim, M., Lee, J., Kang, K., Hong, Y., Bang., S.: Re-engineering Software Architecture of Home Service Robots: A Case Study, International Conference on Software Engineering, Missouri, USA, pp.505-513, 2005.
[10] Medvidovic, N., Taylor, R. N.: Exploiting architectural style to develop a family of applications, Software Engineering. IEE Proceeding, Vol.144, No.5-6. October/December, 1997.
[11] Lago, P., Vliet, H.: Observations from the Recovery of a Software Product Family, Software Product Line Conference 2004, LNCS Vol 3154 Springer-Verlag, Berlin Heidelberg, New York , 2004.
[12] Basset, P.G.: Framing Software Reuse: Lessons from the Real World. Prentice Hall, Yourdon Press, 1997.
[13] Fowler, M., Beck, K., Brant.,J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
[14] Bosch, J., Hogstrom, M.: Product Instantiation in Software Product Lines: A Case Study. Second International Symposium on Generative and Component-Based Software Engineering, LNCS 2177, 2001.

이 재 준

1991 서강대학교 수학과(이학사)
1998 포항공과대학교 정보통신대학원 정보통신학과(석사)
2006 포항공과대학교 컴퓨터공학과 박사과정
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
E-mail : gibman@postech.ac.kr

김 병 길

2004 연세대학교 컴퓨터산업공학과 (공학사)
2006 포항공과대학교 컴퓨터공학과 (공학석사)
2006 LG 소프트웨어 공학 연구소
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
E-mail : dayfly@postech.ac.kr

장 엽

2005 고려대학교 컴퓨터학과(공학사)
2006 포항공과대학교 컴퓨터공학과 석사과정
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
E-mail : ranivris@postech.ac.kr

강 교 철

1973 고려대학교 통계학과(이학사)
1974 콜로라도대학교 산업공학과(석사)
1982 미시간대학교 소프트웨어공학(박사)
2006 포항공과대학교 컴퓨터공학과 정교수
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
E-mail : kck@postech.ac.kr