# 프로덕트 라인 엔지니어링에서 동적 재구성이 가능한 임베디드 시스템 개발을 위한 휘처 중심의 방법

특집 08

목 차



이재준 · 김경석 · 고재훈 · 강교철
(Fraunhofer IESE · 삼성전자 · 포항공과대학교)

## 요 약

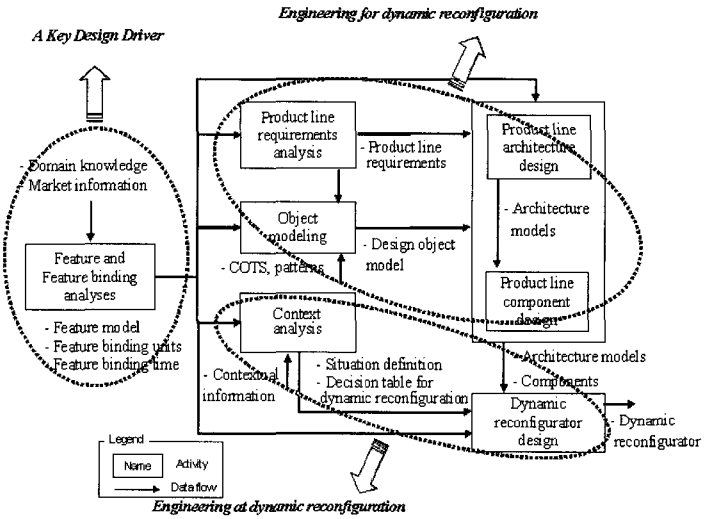
동적 제품 재구성은 런타임 시에 제품의 구성을 변경하는 것을 의미한다. 최근에는 여러가지 응용 분야에서 동적 제품 재구성에 대한 요구가 늘어나고 있으며, 특히 다가오는 유비쿼터스 환경하에서 임베디드 시스템의 동적 재구성에 대한 수요가 증가할 것으로 예상된다. 하지만 대부분의 프로덕트 라인 엔지니어링 방법은 정적 구성 제품을 위한 재사용 가능한 핵심 자산을 개발하는데 초점을 맞추고 있다. 본 논문에서는 휘처를 기반으로한 동적 재구성이 가능한 핵심 자산을 개발하는 접근법을 제안한다. 이 접근법은 동적 재구성 가능한 제품의 변경점을 찾아내고 관리하기 위하여, 휘처 바인딩 분석 결과를 중요한 디자인 드라이버로서 사용한다. 또한 런타임 시에 제품의 재구성을 감시하고 관리하는 재구성기를 위한 개념적인 모델을 제공하며, 이 모델을 바탕으로 전역 구성기의 설계 방법을 제시한다. 이 모델들은 홈 서비스 로봇 프로덕트 라인의 예제를 통하여 설명된다.

## 1. 서 론

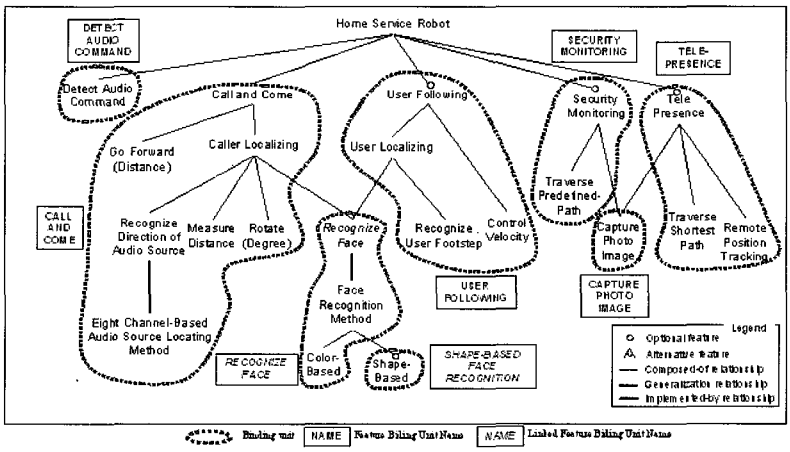
다가오는 유비쿼터스 환경에서는 주변 환경의 변화나 사용자의 요구 사항의 변화에 따라 알맞은 제품의 변경 기능이 요구된다. 특히 제품에 내장된 임베디드 소프트웨어는 동적으로 아키텍처를 재구성하고, 사전에 개발된 여러 기능 요소들을 선택, 탑재할 수 있어야 한다. 이 기능을 통해 한정된 에너지의 보다 효율적인 사용 및 소프트웨어 적시성의 향상을 기대할 수 있다.

프로덕트 라인 엔지니어링은 소프트웨어의 재사용의 한 패러다임으로서, 향상된 품질의 제품군을 보다 짧은 기간에 개발하는 것을 목표로 한다[1,2,3]. 하지만, 대부분의 프로덕트 라인 엔지니어링은 정적 구성 제품의 개발에만 초점을 맞추고 있다[4,5]. 즉, 모든 변경 가능한 사항은 사용자들에게 제품이 전달되기 전에 결정되며 사용자들은 결정된 사항을 바꾸기 어렵다.

동적 제품 재구성은 사용되는 제품의 구성을 런타임시에 변경하는 것을 의미한다. 제품의 휘




(그림 1) 접근 방법의 주요 활동




(그림 2) HSR 프로덕트 라인의 휘처 모델과 바인딩 유닛 식별

〈표 1〉 HSR 프로덕트 라인을 위한 휘처 모델 및 바인딩 유닛

| 제품 휘처 | 설명 |
|---|---|
| Call and Come (CC) | 로봇은 마이크를 통해서 얻어진 소리의 세기를 비교하여 소리가 들려오는 방향이 어디인지를 판단한다. 그 후 소리가 들려오는 방향으로 회전한 뒤 전방 카메라를 통해서 얻어진 영상 데이터를 통해 사용자의 얼굴을 인지하려고 노력한다. 만약 사용자의 얼굴이 인식되면 로봇은 사용자로부터의 거리가 1m이하가 될 때까지 전진한다. |
| User Following (UF) | UF서비스가 활동되면 로봇은 계속하여 영상 데이터와 구조적 관점에로부터 얻어진 데이터를 이용하여 유저의 현재 위치를 확인하고 노력한다. 로봇은 사용자로부터의 거리를 1m이하로 유지한다. "Stop"이라는 명령을 UF서비스를 종료시킨다. |
| Security Monitoring (SM) | 로봇은 자도를 이용하여 집 주변에 대한 감시 기능을 수행한다. 이 서비스는 로봇이 어딘속에 혼자 남겨졌을 때 자동으로 발동된다. 침입이나 사고는 영상과 음성 데이터로서 얻지 될 수 있는 이벤트로 정의된다. 그러한 이벤트가 감지되면 로봇은 영상 이미지를 저장한다. 부가적으로 로봇은 사용자에게 메시지를 전송하여 이벤트를 보고한다. |
| Tele-Presence (TP) | 원거리의 사용자는 PDA를 이용하여 로봇을 제어할 수 있다. 사용자는 로봇에게 PDA에 표시되는 지도의 특정한 위치로 이동을 명령할 수 있다. |

처를 동적으로 추가, 삭제, 변경하거나 아키텍처의 구조를 변경[5,6]하는 것이 그 예이다. 동적 제품 재구성은 자기 치유 시스템[7,8,9], 컨텍스트 인식 컴퓨팅[10,11], 소프트웨어 컴포넌트 배치[12,13,14]나 유비쿼터스 컴퓨팅[15,16] 등 여러 가지 분야에서 연구 되어 왔다. 하지만 지금까지

의 연구에서는 단일 제품에 관한 동적 재구성이 연구되어 왔을 뿐, 제품 계열에 대한 동적 재구성 연구는 다루어지지 않았다.

동적 재구성 제품은 다음과 같은 사항을 만족하여야 한다.

- 제품의 현재 상황을 감시할 수 있어야 한다. (예시: 수행 상황)
- 재구성에 따른 영향과 사용 가능한 자원을 고려하여 재구성 요청을 평가해야 한다.
- 재구성시 현재 활성화된 서비스에 대한 처리 전략이 수립되어 있어야 한다.
- 시스템의 무결성을 유지하면서 동적 재구성이 이루어져야 한다.

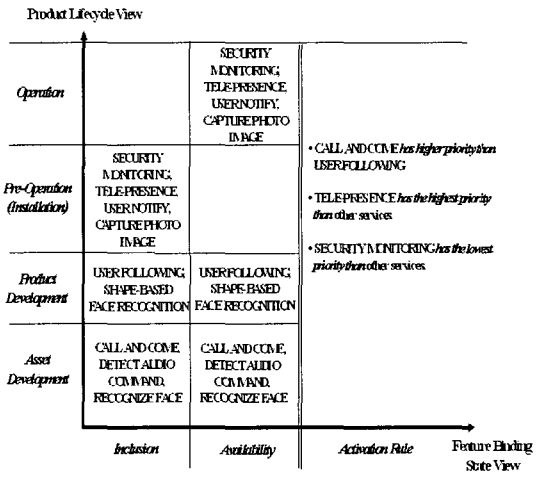
또한 프로덕트 라인 엔지니어링에서는 프로덕트 라인의 변화를 동적으로 다룰 수 있는 능력이 필요하다. 예를 들어서 홈 서비스 로봇이 컨텍스트에 대응되는 서비스를 최적으로 제공하기 위해 동적으로 재구성되어야 한다고 가정하자. 이 경우 고급 홈 서비스 로봇은 고가의 Security Monitoring 휘처를 항상 사용할 수 있지만, 보급형 홈 서비스 로봇의 경우에는 런타임시에 사용 가능 여부가 결정될 수 있을 것이다. 따라서 고급형 제품에는 개발 단계에서 Security Monitoring 휘처가 결합되나, 보급형 제품에는 런타임시에 결합될 수 있을 것이다. 그러므로 동적 재구성을 지원하는 핵심 자산을 개발하기 위해서는 컨텍스트에 민감한 제품 서비스와 제품 실행의 동적 변화를 체계적으로 관리할 수 있어야 한다. 본 논문에서는 동적 재구성이 가능한 핵심 자산을 개발하기 위한 체계적인 접근방법과, 런타임시에 제품의 구성을 감시하고 관리할 수 있는 재구성기를 제안한다. 여기서는 휘처와 바인딩 시점 관점에서 프로덕트 라인을 분석하며, 분석 결과를 바탕으로 핵심 자산을 개발한다. (그림 1 참고) 그 뒤, 재구성 컨텍스트 (언제 재구성 될 것인가), 재구성 전략(어떻게 재구성 할 것인가), 재구성 동작(재구성하기 위해 무엇을 할 것인가)을 고려하여 재구성기를 개발한다. 본 논문에서는 이 방법을 홈 서비스 로봇 (HSR) 제어 소프트웨어에 적용해 보았다.

## 2. 동적 재구성을 위한 엔지니어링

이 절에서는 동적 재구성을 위한 프로덕트 라인 핵심 자산을 개발하는 활동에 대해서 소개한다.

### 2.1 휘처와 휘처 바인딩 분석

휘처 분석 단계에서는 프로덕트 라인에서 제품들의 외부로 보이는 특징들을 찾아내어 휘처 모델을 구성한다. (그림 2의 HSR의 휘처모델을 참고) 구성된 휘처 모델은 휘처 바인딩 분석을 통해 더욱 정제되는데, 이 단계에서는 휘처 바인딩 유닛 식별과 휘처 바인딩 시점 결정이 수행된다. 휘처 바인딩 유닛 식별은 서비스 휘처를 식별하는 것부터 시작되는데, 이는 시스템의 주된 기능을 나타내며, 추가/삭제의 단위가 된다. (HSR의 서비스 휘처에 대한 예는 <표 1>을 참고 [17] 위의 휘처 모델에서는 CC, UF, SM 그리고 TP 휘처가 서비스 휘처로 분류될 수 있음) 이 서비스 휘처로부터 시작하여 휘처들 사이의 관계를 따라 각 휘처를 검토하여 휘처 바인딩 유닛에 포함 가능한지 여부를 결정한다.




(그림 3) HSR 바인딩 유닛의 바인딩 시점

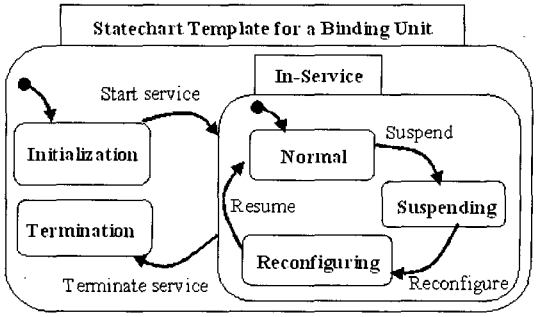
휘처 바인딩 유닛이 식별되면 그것의 바인딩 시점을 결정하게 된다. (그림 3 참조) 이는 두 가지 관점에 기반하여 분석되는데 제품 주기 관점과 바인딩 상태 관점이 그것이다. 제품 주기 관점은 어떤 휘처가 어느 단계에서 제품에 포함되는지를 나타낸 것이고 바인딩 상태 관점은 휘처의 포함, 사용가능성 그리고 활성화 규칙을 나타낸 것이다. (휘처가 물리적으로 제품에 포함되어 있어도 사용가능하지 않을 수 있다.)

휘처 바인딩 분석을 통하여 런타임시에 결합되어야 하는 휘처들을 명시적으로 파악할 수 있다. 다음 절에서는 서비스와 재구성 행위 명세에 대해서 설명한다.

## 2.2 휘처 바인딩 유닛의 행위와 기능적 명세

동적 재구성이 가능한 제품은 현재 제품 구성에 대한 변화 요청을 올바르게 처리할 수 있어야 한다. 본 논문에서는 동적 재구성 요청 처리에 사용되는 공통 행위를 정의한 스테이트차트 템플릿을 제안한다. (그림 4의 템플릿을 참고) 최상위 레벨 스테이트차트 명세는 Initialization, Termination, In-Service라는 세가지 상태를 가지고 있다. Initialization과 Termination 상태는 바인딩 유닛의 초기화와 종료 시점에 필요한 작업들의 명세를 위한 것이다. (예시: 장치의 초기화와 종료) In-Service 상태는 Normal, Suspending, Reconfiguring 상태로 정제되는데 Normal 상태는 바인딩 유닛의 행위를 명세하고, Suspending 상태는 동적 재구성에 필요한 사전 작업들을 포함하며 (예시: 현재 상태 저장, 활성화 서비스 끝내기 등), Reconfiguring 상태에서는 실제 재구성을 수행한다.

스테이트차트 템플릿은 각각의 휘처 바인딩 유닛에 맞추어 정제된다. 예를 들어서 SM의 Normal 상태는 선택적 바인딩 유닛인 USER NOTIFY이 선택되었는지 여부에 따라 두 가지가 있을 수 있다. 실제 재구성시 해당 바인딩 유닛의 선택 여부에 따라 서로 다른 스테이트 차트가 선택된다.

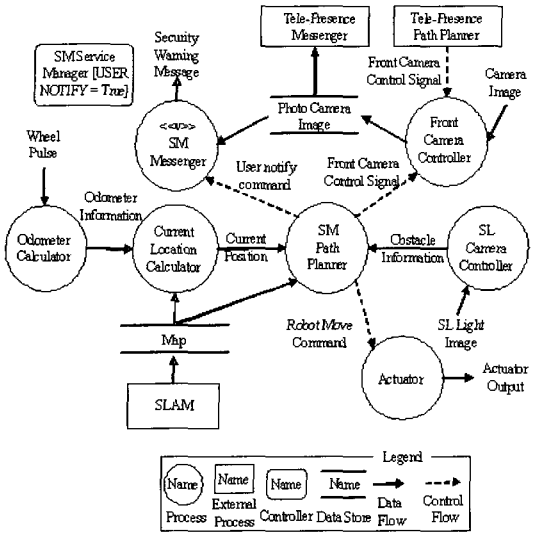



(그림 4) 휘처 바인딩 유닛의 행위 명세를 위한 스테이트차트 템플릿

Suspending 상태에는 현재의 활성화 서비스를 어떻게 다룰 것인가에 대한 전략이 정의되며, 다음 중 하나로 선택된다. 1)현재 상태를 저장하지 않고 Suspending 상태로 바뀐다. 2)상태 회복을 위해서 현재 상태를 저장하고 Suspending 상태로 바뀐다. 3)현재 활성화 서비스를 마치고 Suspending 상태로 들어간다. 4)Suspending 상태로 들어가지 않는다. (예시: 재구성 도중 서비스의 중단없는 제공) 예를 들어 UF는 자신의 서비스 행위를 Suspending 상태와 Reconfiguring 상태로 복제함으로써 재구성시 서비스를 중단없이 제공하도록 명세되어있다.

휘처 바인딩 유닛의 행위가 명세된 후에는 데이터 플로우 다이어그램(DFD)을 이용하여 휘처 바인딩 유닛의 기능적인 요소를 명세한다. DFD는 프로세스 간의 데이터와 제어 신호의 흐름을 나타내며, 각 프로세스들은 스테이트 차트 명세에 의하여 제어된다. 예를 들어 그림 5 에 묘사되어 있는 프로세스는 SM Service Manager 스테이트차트에 의해서 제어된다.

DFD안의 변경점을 나타내기 위해서 <<v>>라는 스테레오타입이 사용된다. 만약에 <<v>>표시가 되어있는 바인딩 유닛이 선택되지 않는다면 DFD에서 해당 데이터와 제어 신호들이 제거된다.




(그림 5) SM을 위한 DFD 명세

휘처 바인딩 유닛간에 공유하는 프로세스나 데이터 또한 기능 분석을 통해 식별되어야 한다. 예를 들어서 (그림 5)의 Tele-Presence Path Planner 는 Front Camera Controller 프로세스를 공유한다. 이러한 공통 기능에 대한 정보는 프로덕트 라인 아키텍처와 컴포넌트를 디자인하는데 중요한 정보가 된다: 프로세스나 데이터를 공유하는 휘처 바인딩 유닛의 동적인 제거나 추가는 다른 휘처 바인딩 유닛에 영향을 미칠 수 있다. 예를 들어서 SM의 제거를 위한 동적 재구성이 일어날 때 TP와 공유하고있는 Front Camera Controller 프로세스가 제거되어서는 안된다. 공유 프로세스와 데이터의 처리에 관한 결정은 자원 사용, 동시성등을 고려하여 디자인 단계에서 이루어진다.

다음 절에서는 휘처 바인딩 유닛이 중요한 드라이버로 사용된 프로덕트 라인 아키텍처와 컴포넌트의 디자인이 소개된다.

## 2.3 프로덕트 라인 아키텍처와 컴포넌트 디자인

### 2.3.1 동적으로 재구성이 가능한 아키텍처 디자인을 위한 엔지니어링 가이드라인

이 절에서는 아키텍처 디자인을 위한 5개의 가이드라인을 제시한다. 이러한 가이드라인의 목적은 동적 재구성의 가시성과 추적성을 높이기 위함이다.

가이드라인 1: 제어 컴포넌트와 데이터 컴포넌트의 분리. 휘처 바인딩 유닛의 행위와 데이터흐름은 각각 스테이트차트와 DFD를 통해서 명세된다. 따라서 행위의 분석은 제어와 관련되고 데이터흐름의 분석은 기능과 관련이 된다. 이러한 제어와 기능의 분리는 휘처 바인딩 유닛과 아키텍처 컴포넌트 사이의 대응 관계를 보다 명확하게 한다.

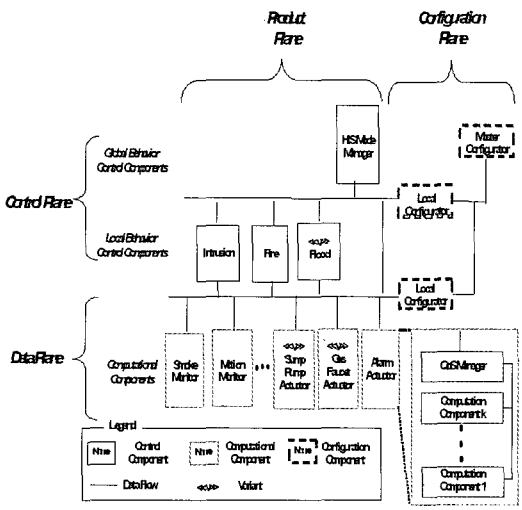
가이드라인 2: 전역 행위와 지역 행위의 분리. 각 휘처 바인딩 유닛들은 서비스를 제공하기 위한 고유의 상태(지역 행위)를 가지고 있고, 이들은 지역 행위 제어 컴포넌트에 할당된다. 시스템 전체의 무결성을 보장하기 위해서는 전역 행위(예시: 안전에 직결된 명령은 현재 활성화된 서비스를 무시하고 실행되어야 한다.)가 정의되어야 하고, 지역 행위 제어 컴포넌트들은 전역 행위(예시: 시스템 모드)와 함께 조율되어야 한다.

가이드라인 3: 휘처 바인딩 유닛과 일치하는 아키텍처 컴포넌트의 발견. 올바른 크기를 가진 올바른 아키텍처 컴포넌트를 찾는 일은 매우 중요하다. 여기서는 휘처 바인딩 유닛을 기반으로 아키텍처 컴포넌트를 찾고 컴포넌트와 휘처 바인딩 유닛을 명시적으로 대응시키는 방법을 사용한다. 이러한 방법은 재구성에 따른 영향을 줄여주며 서로 다른 구성들 사이의 추적성을 향상시켜 준다.

가이드라인 4: 계산을 담당하는 컴포넌트 에서 관리 정책의 분리. 앞서 언급했듯이 재구성후의 시스템의 무결성을 보장하기 위해서는 휘처 바인딩 유닛들의 공통적인 기능요소들을 주의깊게 관리해야 한다. 따라서 데이터 플레인에 QoS 컴포넌트를 도입하여 공통적인 기능에 대한 관리 정책을 명시적으로 명세할 수 있게 한다.

가이드라인 5: 컨텍스트와 서비스의 분리. [5], [6]에서 강조된바와 같이, 서비스 관점과 제품 재구성 관점의 분리는 동적 재구성의 복잡도를 줄이기 위해 중요하다. 즉 제품 재구성 관점은 현재 제품의 상황과 재구성 트랜젝션을 감시하는데 집중하는 반면, 제품 서비스 관점은 서비스를 제공하기 위한 컴포넌트의 계산과 상호작용에 초점을 맞추어야 한다.

### 2.3.2 HSR 프로덕트 라인의 아키텍처 모델




(그림 6) HSR 프로덕트 라인을 위한 아키텍처 모델

우리는 (그림 6)에 나타난 HSR 아키텍처 모델을 개발하는데 C2 스타일의 아키텍처를 적용하였다[19,20]. C2 스타일은 '브릭'이라고 불리는 컴포넌트와 계층 구조를 통해서 유연한 아키텍처 스타일을 제공한다. 브릭은 위/아래 포트와 포트를 연결하는 버스 스타일의 커넥터를 통해서 다른 브릭들과 메시지를 주고 받을 수 있다.

C2 스타일은 변화에 대해서 유연하지만 그 스타일 자체는 어떠한 디자인 가이드 라인도 제시하지 않는다. 따라서 런타임 유연성을 제공하기 위하여 C2 스타일을 다음과 같이 정제하고 확장하였다. 일단 앞에서 제시한 엔지니어링 가이드라인을 적용하여 C2 스타일을 두개의 플레인으로 나누었다. 먼저 아키텍처 모델을 구성 플레인과 제품 플레인으로 나누었고, 제품 플레인은 다시 제어 플레인과 데이터 플레인으로 나누었다. 각 플레인들에 대한 자세한 설명은 다음과 같다.

제품 플레인의 제어 플레인(관련 가이드라인: 1, 2, 3): 제어 플레인의 컴포넌트들은 시스템의 행위를 제어한다. 각 지역 행위 제어 컴포넌트(예: CC, UF, SM, 과 TP)는 휘처 바인딩 유닛의 행위를 정의하는데, 각각은 독립적으로 실행되고 테스트 될 수 있다. 전역 행위 제어 컴포넌트들은 시스템 모드(예: 초기화, 중단, 전력 절약 모드)와 지역 제어 컴포넌트 사이의 상호 작용 정책(예: 우선순위, 동시성)을 결정한다.

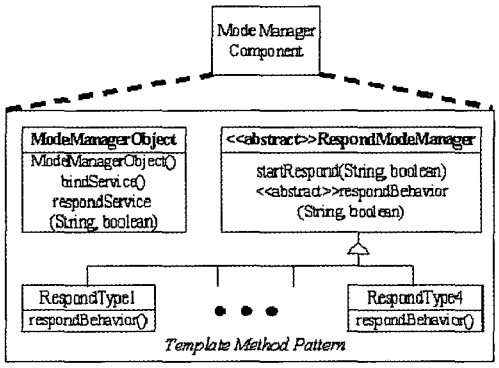
제품 플레인의 데이터 플레인(관련 가이드 라인: 1, 4): 데이터 플레인은 센서에서 얻은 데이터를 처리하여 이벤트나 임시 데이터를 만들어내는 계산 브릭들로 이루어져 있다. HSR Mode Manager 브릭으로 보내지는 이벤트 데이터는 시스템 전체 상태를 변경하는데 사용되는 반면 다른 계산 브릭들로 보내지는 임시 데이터는 해당 브릭의 입력 데이터로 사용된다. 각 계산 브릭의 기능은 휘처 바인딩 유닛의 DFD 에 명세되어 있는데 이는 휘처 바인딩 유닛과 브릭 사이의 관계를 명시적으로 나타낼 수 있다는 것을 의미한다. 따라서 휘처 바인딩 유닛의 추가/제거로 생기는 변화는 데이터 플레인에서도 추적이 가능하다.

구성 플레인(관련된 가이드라인: 5): 구성 플레인은 컨텍스트 변화의 감지, 재구성 전략의 결정 및 유효성 확인, 재구성 실행을 담당한다. 구성 플레인은 크게 전역 구성기와, 지역 구성기라는 두 가지 타입의 컴포넌트로 구성되어 있다. 전역 구성기는 지역 구성기와 외부 검사기로부터 정보를 모아 컨텍스트 변화를 감지하며, 재구성이 요구될 경우 재구성 수행을 위해서 관련된 트랜잭션들을 처리한다. 반면 지역 구성기는 커넥터에 연결되어 있고 브릭들 사이의 메시지를 감시한다.

아키텍처 디자인에 관한 가이드라인들을 적용함으로써 휘처 바인딩 유닛과 아키텍처 컴포넌트 사이의 관계가 쉽고 명확하게 수립될 수 있었고 휘처 바인딩 유닛 사이의 상호 작용을 눈으로 보고 관리할 수 있게 되었다. 또한 제품의 서비스와 재구성을 분리하여 다룸으로써 각 컴포넌트의 역할이 더욱 명확해지고 컴포넌트 행위 명세의 복잡도를 낮출 수 있었다.

다음 절에서는 프로덕트 라인 컴포넌트 개발을 설명한다.

#### 2.3.3 프로덕트 라인 컴포넌트 개발




(그림 7) Mode Manager의 컴포넌트 명세

프로덕트 라인 컴포넌트의 개발에 있어 주요한 입력물은 휘처 모델, 휘처 바인딩 유닛과 바인딩 시점, 아키텍처 모델들, 디자인 객체 모델이다. 휘처 바인딩 유닛의 동적 재구성을 위해서는 각 바인딩 유닛과 관련된 변경점들이 디자인 객체 모델에서 표현되어야 하고 적절한 바인딩 테크닉을 사용하여 구현되어야 한다. 객체, 메뉴, 플러그인들의 동적 바인딩은 컴포넌트의 동적 바인딩을 보조하는 테크닉들이다.

또한 재구성의 영향을 주의깊게 분석할 필요가 있는데 예를 들어 HSR Mode Manager의 행위는 새로운 프로덕트 구성에 맞추어서 변해야 한다. (그림 7)에 나와 있듯이 우리는 Template Method 패턴을 적용하여 HSR Mode Manager의 행위의 동적 변화를 명세하였다(그림 7은 컴포넌트 명세를 나타낸다). HSR Mode Manager는 선택적 휘처의 조합에 따라서 4개의 서로 다른 행위 명세를 가진다 (예: TP와 SM의 선택여부). 전역 구성기가 제품 재구성을 런타임시에 결정한 후 서비스 휘처들 사이의 상호작용을 다루기 위하여, 적절한 행위 제어 컴포넌트가 HSR Mode Manager에 결합되어야 한다.

이 절에서 우리는 프로덕트 라인을 어떻게 분석하고 핵심 자산들이 어떠한 엔지니어링 프로세스를 거쳐서 개발되는지를 설명하였다. 다음 절에서는 재구성기를 개발하는데 필요한 활동에 대하여 설명한다.

## 3. 동적 재구성 엔지니어링

재구성 과정에 걸쳐 시스템의 무결성은 보장되어야 한다. 재구성된 시스템은 심각한 오류를 발생시키거나 부정확한 행위를 보일 수 있는데 이는 1)재구성이 적절하지 않은 상황에서 발동되었을 때 (재구성 시점), 2)부적절한 재구성 전략이 선택되었을 때 (재구성 전략), 3)변경되지 말아야 할 부분이 재구성 트랜잭션에 의해서 변경되었을 때 (재구성 대상)이다. 따라서 이러한 재구성의 세가지 측면(언제, 어떻게, 무엇을)은 정확히 분석되고 명세되어야 한다.

이 절에서는 컨텍스트 정보, 재구성 전략, 재구성 동작을 분석하고 명세하는 방법을 소개한다. 또한 동적 재구성기의 개념적인 디자인도 소개한다.

### 3.1 컨텍스트 분석

<표 2> 컨텍스트 인자 정의

| Attributes / Contextual Parameters | Type (unit of value) | Sampling Rate | Validity |
|---|---|---|---|
| Brightness (B) | Integer (lux) | 10 seconds | 0<= B < 100 |
| Battery Remaining Time (BRT) | Integer (minute) | 10 seconds | 0 < BRT < 300 |

우리의 접근방법에서는, 수행상황을 분석함으로써 재구성 시작 시점을 결정한다. 수행 상황 분석은 컨텍스트 인자 발견, 상황 정의, 그리고 각 상황과 재구성 요구 사이의 연결정의로 이루어진다.

컨텍스트 분석은 시스템의 컨텍스트에 관한 정보를 가지고 있는 환경 요소인 컨텍스트 인자를 발견하는 것으로부터 시작된다. (예: 현재 위치의 밝기, 배터리 잔량). 컨텍스트 인자가 발견된 후에는 각 컨텍스트 인자의 어트리뷰트를 정의하는데 어트리뷰트는 데이터 타입일수도 있고 샘플링 레이트 또는 유효 조건일 수도 있다. (HSR에서의 예제는 표 2를 참조) 타입 열에서는 컨텍스트 인자의 값의 타입이 단위와 함께 정의되며, 샘플링 레이트는 각 인자가 얼마나 자주 검사되는지를 나타낸다. 각 컨텍스트 인자는 그 값이 특정한 구간안에 있거나 미리 정해진 값에 해당될 때만 유효할 수도 있다. 이러한 컨텍스트 인자들의 유효조건은 실제 컨텍스트 변화를 감지하기 위해 사용되기 전에 만족 여부가 검사되어야 한다.

각 컨텍스트 인자가 정의되었다면 특정 상황은 동적 인자의 논리적 표현으로 명세된다. 각 상황은 동적 재구성을 발동시키는 이벤트이기도 하는데 예를 들면 '어둠 속의 로봇'이라는 상황은 밝기(B)가 5보다 작고 배터리 잔량(BRT)가 30보다 크고 SM서비스가 가능할 때로 정의될 수 있다. 이러한 상황에서는 Blind SM and Start Surveillance라는 재구성 요구가 발동되어야 하고 이 재구성 요구에 대한 동적 재구성 전략이 분석되고 명세되어야 한다.

### 3.2 동적 재구성 전략과 동작 명세

동적 재구성 전략은 어떻게 동적 재구성을 실행할 것인가에 관한 것이다. 이는 바인딩 의존성(예: 필요와 배척), 다른 바인딩 유닛에 대한 영향, 필요한 자원(예: 컴포넌트)을 고려하여 명세된다. 각 재구성 전략은 다음의 재구성 6단계를 통하여 명세된다: 1)선조건의 검사, 2)현재 활동중이면서 재구성과 관련된 바인딩 유닛들에게 Suspend 이벤트를 보냄, 3)제거되거나 변경되어야 되는 바인딩 유닛을 제거하거나 매개변수화, 4)새로 더해지는 바인딩 유닛의 생성, 5)후조건 검사, 6)중지된 바인딩 유닛들의 활동을 재개하고 새로 더해진 바인딩 유닛을 시작. 첫번째와 다섯번째 단계는 각각 재구성의 선/후 조건을 검사하는데 이는 필요나 배척 관계에 있는 바인딩 유닛들을 검사함으로써 이루어진다.

두번째 단계에서는 재구성 과정에서 관련된 바인딩 유닛이 일시 중단되어야 하는지 아니면 계속 서비스를 제공해야 하는지를 판단한다. 세번째 단계에서는 더 이상 필요하지 않은 바인딩 유닛을 제거한다. 이 때 바인딩 유닛의 행위가 매개변수를 통해서 변경될 수 있다면 새구성에 맞는 적절한 매개변수가 바인딩 유닛에 보내져야 한다. 네번째 단계에서는 새로이 더해지는 바인딩 유닛이 생성되고 제품에 포함된다. 마지막으로 재구성의 후조건을 검사한후에 바인딩 유닛들을 활성화 한다.

```
dynamicReconfiguration_BindSMandStartSurveillance{
...
invoke_methods checkCurrentArchitectureConfiguration(SM);
...
invoke_methods setSuspendedCC;
invoke_methods setSuspendedUF;
...
add
    condition IsInstantiated(SecurityMonitoring) == YES;
    condition IsInstantiated(FrontCamera) == YES;
...
    weld (topConnector, SecurityMonitoring)
    weld (SecurityMonitoring, bottomConnector))
...
}
```

(그림 8) Bind SM and Start Surveillance를 위한 재구성 동작 명세

재구성 동작 명세는 추상적인 재구성 전략의 이벤트들을 컴포넌트의 행위와 구성을 제어하기

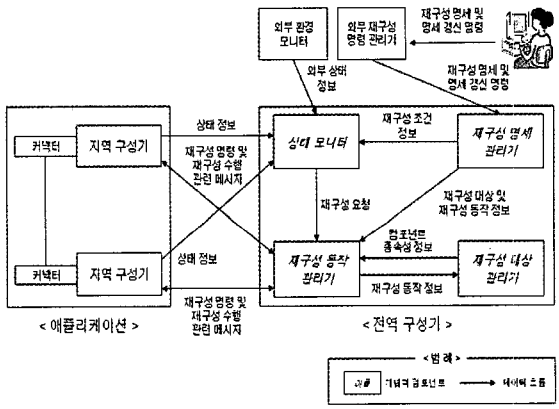
위하여 컴포넌트로 보내질 실질적인 명령들로 정제한다. 예를 들어 새로 생성된 아키텍처 컴포넌트는 'weld' 명령을 사용하여 아키텍처와 연결되는데 이는 특정 컴포넌트를 C2스타일의 커넥터에 연결하는 명령어이다. (그림 8은 bind SM and Start Surveillance 요구에 대한 재구성 동작 명세의 부분을 보여준다.) 다음 절에서는 재구성기의 개념적 모델을 소개한다.

### 3.3 재구성기의 개념적 모델

본 절에서는 재구성기를 디자인하기 위해 (그림 9)의 개념적 모델을 제안한다. 개념 모델은 전역 구성기와 지역 구성기 두 부분으로 이루어져 있다. 전역 구성기는 컨텍스트와 제품의 상태를 감시하고 재구성 요구를 처리하며, 지역 구성기는 각 커넥터의 메시지를 분석하여 전역 구성기에 제품의 상태정보를 제공해 주고 전역 구성기로부터 받은 재구성 명령을 실행한다.

전역 구성기는 동적 재구성 아키텍처 스타일의 핵심 컴포넌트로써, 현재의 아키텍처 구성 정보 관리, 재구성 유발 상황 및 조건 모니터링, 그리고 재구성 유발 조건 발생 시 재구성 작업을 수행한다. 그림 9는 전역 구성기와 타 구성요소와의 데이터 흐름 내역과 전역 구성기의 내부 컴포넌트에 대한 개념도이다. 내부 컴포넌트에 대한 설명은 다음과 같다.

- 재구성 명세 관리기: 재구성 관리자는 외부 재구성 명령 관리기를 사용하여 재구성 명세를 추가 또는 삭제한다. 재구성 명세 추가 시 재구성 명세 관리기는 명세를 분석하여 재구성 발생 조건과 관련된 정보는 상태 모니터에게, 재구성 대상 및 재구성 동작과 관련된 정보는 재구성 동작 관리기에게 전달한다. 삭제 시에는 상태 모니터가 재구성 발생 조건과 관련된 정보를 삭제하도록 하며 재구성 동작 관리기가 재구성 대상 및 동작 관련 정보를 삭제하도록 한다.
- 상태 모니터: 제품의 내·외부 상태를 모니터링하는 컴포넌트로써, 외부 상황은 외부 환경 모니터로부터, 내부 상황은 지역 구성기로부터 수집한다. 모니터링 결과 재구성 조건을 만족시키면 재구성 동작 관리기가 재구성 동작을 수행하도록 한다.
- 재구성 대상 관리기: 현재의 아키텍처 구성 상태를 관리하며 재구성 후에는 이 정보를 갱신한다. 재구성 동작 관리기가 이 정보를 이용하여 각 컴포넌트의 상호 작용 상태를 분석 할 수 있도록 한다.
- 재구성 동작 관리기: 초기 아키텍처를 구성하며, 동작 중 상태 모니터가 재구성 수행을 요청하면 재구성을 실행한다.




(그림 9) 재구성기의 개념적 모델

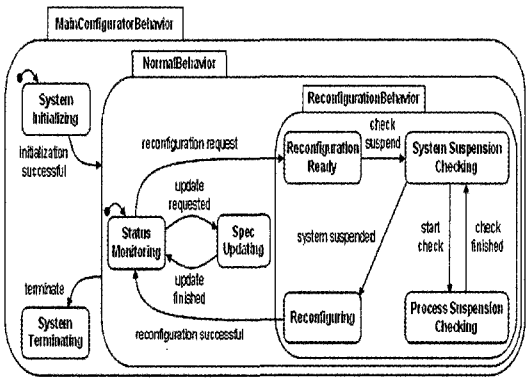
## 4. 전역 구성기 설계 기술

이 절에서는 앞 절의 재구성기 개념적 모델을 바탕으로 전역 구성기의 설계에 사용 할 수 있는 구체적인 기술을 제시한다. 즉, 재구성 기술을 지원하기 위한 전역 구성기의 행위 명세, 재구성 대상 관리기와 재구성 동작 관리기의 역할, 그리고 재구성 동작 관리기가 프로세스의 중지 가능성을 판단하는 방법이 구체적인 명세나 알고리즘과 함께 제시된다.

여기서는 동적 재구성 단계의 첫번째에 해당하는 선조건 검사 중 프로세스 사이의 의존 관계에 대한 검사 방법을 설명한다. 하나 이상의 휘처

바인딩 유닛이 하나의 프로세스를 구성한다. 예를 들어 CC 훼처 바인딩 유닛과 RF 훼처 바인딩 유닛으로 구성된 프로세스가 생성되어 CALL AND COME 명령을 수행할 수 있을것이다. 그러므로 이 절에서는 훼처 바인딩 유닛의 집합으로 구성된 프로세스를 단위로 설명한다.

## 4.1 전역 구성기 행위 명세




(그림 10) 전역 구성기 행위 명세

위의 (그림 10)은 전역 구성기의 행위 명세이다. 이 중 본 절에서 다룰 Reconfiguration Behavior 상태의 하위 상태에 대한 설명은 다음과 같다.

- Reconfiguration Ready: 재구성을 준비하기 위해 모든 프로세스들에게 재구성 준비 명령 메시지를 출력
- System Suspension Checking: 재구성을 시작하기 전에 모든 프로세스들이 중지되었는지 확인
- Process Suspension Checking: 프로세스가 재구성 가능 상태에 도달할 경우 그 프로세스의 중지 가능성을 분석
- Reconfiguring: 재구성 명령을 수행하며 프로세스의 입/출력 의존 관계 정보를 갱신

다음 절에서는 ReconfigurationBehavior 상태를 기반으로 재구성 대상 관리기 및 재구성 동작 관리기에 대해 보다 자세하게 설명한다. 여기서는 재구성 전략 중 모든 프로세스들이 안정 상태인 재구성 가능 상태에서 정지된 후 재구성이 시작되며, 재구성 완료 후 일괄적으로 모든 프로세스들을 초기 상태에서 재시작시키는 방식을 사용한다.

## 4.2 재구성 대상 관리기와 재구성 동작 관리기

전역 구성기의 내부 컴포넌트 중 재구성 대상 관리기와 재구성 동작 관리기에 대해 설명한다.

### 4.2.1 재구성 대상 관리기

재구성 대상 관리기는 현재의 아키텍처 구성 상태를 관리하며, 재구성 후 이 정보를 갱신한다. 여기서는 재구성 대상 관리기가 프로세스들 간의 입/출력 의존 관계 정보를 관리하는 방법을 소개한다.

입/출력 의존 그래프는 다음과 같이 정의할 수 있다.

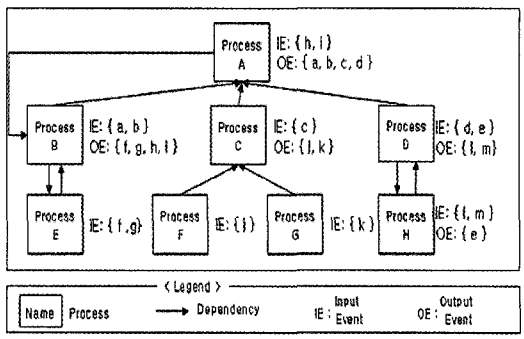
● 입/출력 의존 그래프 DG = < V, D >

- V: 프로세스 집합.
- D: 입/출력 의존 관계 집합.

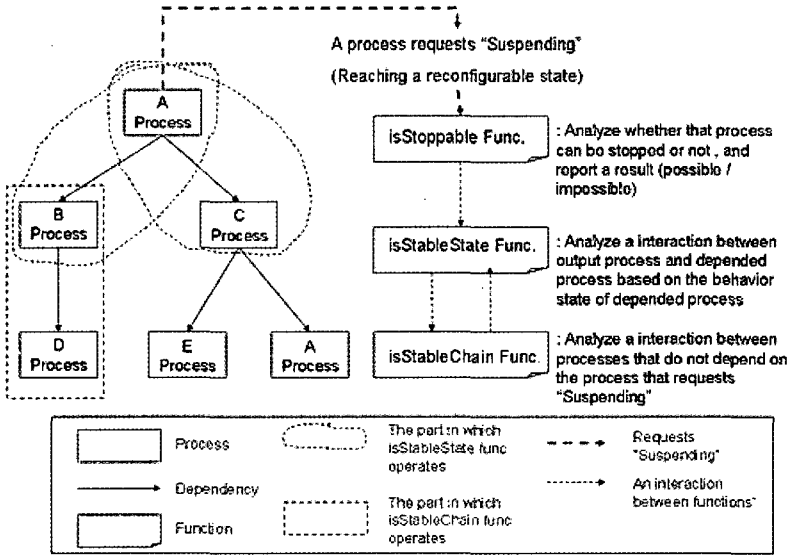
{(pi, pj) | pi, pj ∈ V, pi ≠ pj, IE(pi) ∩ OE(pj) ≠ ø}

- IE(pi): 프로세스 pi의 입력 이벤트 집합.
- OE(pj): 프로세스 pj의 출력 이벤트 집합.

예를 들어, 그림 11에서 IE(H 프로세스) ∩ OE(D 프로세스) = {l, m} 이므로 H 프로세스에서 D 프로세스로의 의존 관계가 있다.




(그림 11) 프로세스 입/출력 의존 그래프 예제




(그림 12) 프로세스 중지 가능성 판단 알고리즘 개요

### 4.2.2 재구성 동작 관리기

재구성 동작 관리기는 제품 초기화 시 초기 아키텍처를 구성하며, 실행 중 상태 모니터로부터 재구성 수행 요청을 수신하면 재구성 동작을 실행한다. 여기서 다루어질 재구성 동작은 재구성 실행 전에 프로세스들 간의 상호 작용을 고려하면서 모든 프로세스들을 중지시키는 과정과 프로세스의 추가나 삭제와 같은 구조적 변경을 수행하고 제품을 재시작 시키는 과정으로 구성된다. 재구성 동작 관리기의 세부 동작 과정은 다음과 같다.

1. 모든 프로세스에게 재구성 준비를 요청 한다.
2. 모든 프로세스를 중지한다.
3. 재구성 가능 상태 도달 메시지 수신시 입/출력 의존 정보를 활용하여 연관 관계가 있는 프로세스의 입/출력 정보를 분석한 후 프로세스의 중지 요청을 수락 또는 거부한다.
4. 모든 프로세스가 중지한 후 제거, 대체, 추가 등 재구성 명령을 수행하며, 재구성 완료 후 모든 프로세스를 재시작 한다.

### 4.2.3 프로세스 중지 가능성 판단 방법

이 절에서는 프로세스가 재구성 가능 상태가 되어 실행 중지를 요청할 때, 재구성 동작 관리기가 수행하는 프로세스 중지 가능성 판단 방법을 알고리즘으로 설명한다. 알고리즘의 개괄적인 구성은 (그림 12)와 같다.

● 실행 중지를 요청한 프로세스의 중지 가능성을 판단하는 알고리즘

```
bool isStoppable(중지 요청한 프로세스Pj 이름 ) {
  int dependency_count = Pj에 의존하는 프로세스 수;
  if (dependency_count == 0) <--
    return true;
  else {
    for(int count = 0; count < dependency_count; count++)  {
      String 프로세스Pk 이름 = Pj에 의존하는 프로세스 이름 목록[count];
      if(Pk는 중지 상태에 있음)
        continue;
      else {
        bool result = isStableState(프로세스Pj 이름, 프로세스 Pj 이름, 프로세스
          Pk 이름);
        if(result == true)
          continue;
        else
          return false;
      } //end else
    } // end for loop
    return true;
  } // end else
} // end algorithm
```

<-- 중지 요청한 프로세스에 의존하는 프로세스의 수가 0일 경우 중지 요청한 프로세스를 중지시킬 수 있다.

● 의존 프로세스의 행위 상태를 기반으로 출력 프로세스와 의존 프로세스의 상호 작용 상태를 분석하는 부분

```
bool isStableState(중지 요청한 프로세스Pj 이름, 출력 프로세스Pm 이름,
의존 프로세스Pn 이름) {
  if(Pn은 재구성 가능 상태에 있지 않음)  {
    if(Pm의 출력 이벤트와 일치한 Pn의 입력 이벤트 중 수신 플래그가 false인
       이벤트가 존재) <--
      return false;
    else if(Pm의 출력 이벤트와 일치한 Pn의 입력 이벤트 중
         "without_rs_cycle_input_events" 에 정의되어 있으며 순환 플래그 값이
         true인 이벤트가 존재)
      return false;
  } //end if(Pm은 재구성 가능 상태에 있지 않음)
  if(Pm의 출력 이벤트와 일치한 Pn의 입력 이벤트 중
  "with_rs_cycle_input_events" 에 정의된 것이 없거나, 정의되어 있다면 순환
  플래그 값이 true인 이벤트가 비 존재)
    return true;
  else {
    bool result = isStableChains(프로세스Pj 이름, 프로세스Pn 이름);
    if(result == true)
      return true;
    else
      return false;
  } // end else
} // end isStableState function
```

<-- 수신 플래그 값이 false라는 의미는 의존 프로세스가 아직 입력 이벤트를 받지 않았다는 뜻이다.

● 중지 요청한 프로세스에 직접적으로 의존하고 있지 않는 프로세스들간의 상호 작용을 분석하는 부분.

```
bool isStableChains(중지 요청 프로세스Pj 이름, 비교 시작 프로세스Px 이름) {
  if(비교 시작 프로세스를 Px로 시작하여 분석한 경험이 없음) <--
  {
    비교 시작 프로세스를 Px로 시작하여 분석한다고 갱신;
    int dependency_count = Px에 의존하는 프로세스 수;
    if(dependency_count == 0)
      return true;
    else {
      for(int count = 0; count < dependency_count; count++) {
        String 프로세스Py 이름 = Px에 의존하는 프로세스 이름 목록[count];
        if(프로세스Py 이름 != 중지 요청 프로세스Pj 이름)  {
          if(Py는 중지 상태에 있음)
            continue;
          else {
            bool result = isStableState(프로세스Pj 이름,
            프로세스 Px 이름, 프로세스 Py 이름) ;
              if(result == true)
                continue;
              else
                return false;
            } // end else
          } // end if(프로세스Py 이름 != 중지 요청 프로세스Pj 이름)
          else
            continue;
        } // end for loop
        return true;
      } // end else
    } // end if(비교 시작 프로세스를 Px로 시작하여 점검한 경험이 없음)
    else
      return true;
  } // end isStableChains function
```

<-- 프로세스들 간의 입/출력 의존 관계에 순환이 존재할 수 있다. 순환이 존재하는 상황에서 재귀 함수를 호출하면 영원히 재귀 함수를 반복 호출하게 된다. 따라서 비교 시작 프로세스를 시작으로 했던 분석 경험이 있다면 이는 순환의 시작을 의미하므로, 이를 피하도록 제어해야 된다.

이 알고리즘에서 한가지 주의해야 될 점은 프로세스들의 입/출력 의존 관계에서 한 프로세스에 여러 프로세스들이 의존하고 있거나 한 프로세스가 여러 프로세스들에 의존하고 있는 경우이다. 이 경우 의존 관계에 있는 프로세스의 모든 입력 이벤트들의 상태를 고려하지 않고 이들 중 실제로 연관된 이벤트만을 비교해야 된다. 실제로 연관된 이벤트는 각 프로세스의 입력 이벤트와 출력 이벤트를 비교하여 결정할 수 있다.

## 5. 토 의

이 절에서는 본 논문에서 제시된 방법의 장, 단점을 토의한다. 본 접근방법은 휘처 바인딩 분석 정보를 중요한 드라이버로 사용하여 동적으로 재구성 가능한 핵심 자산을 만든다. 이러한 휘처 바인딩 기반 방법은 프로덕트 라인 엔지니어에게 다음과 같은 가이드라인을 제시해 줄 수 있다.

### 5.1 적절한 크기의 프로덕트 구성 유닛의 발견

바인딩 유닛은 서비스를 정확히 제공하기 위해서 제품에 같이 포함되어야 하는 휘처들로 구성되어 있다. 따라서 바인딩 유닛을 구성하는 휘처들을 구현하는 모든 컴포넌트들은 휘처 바인딩 유닛이 런타임에 결합될 때 같이 결합되어야 한다. 이러한 그룹화는 제품 구성의 동적인 변화를 다루는데 복잡도를 감소시켜 주며 동적 재구성 요구의 영향을 분석하는데 도움을 준다.

### 5.2 휘처 바인딩 시점의 명시적 표현

휘처 바인딩 시점의 중요성이 널리 인지 되었음에도 불구하고[4, 21, 22] 휘처 바인딩 시점을 분석하고 명세할 때의 난점에 관해서는 명시적으로 다루어지지 않았다. 본 연구에서 휘처 바인딩 시점은 제품 주기 관점과 휘처 바인딩 상태 관점이라는 두가지 관점에서 결정된다. 우리는 이 두가지 관점에서의 바인딩 시점 분석을 통하여 어떠한 휘처들이 동적으로 재구성되는지 정확히 찾을 수 있다.

### 5.3 변경점 의존의 향상된 관리

휘처 바인딩을 기반으로 하는 방법을 사용함으로써 런타임에서 변경점간의 바인딩 의존성이 휘처 바인딩 유닛과의 대응을 통하여 효과적으로 발견되고 관리될 수 있다. 예를 들면 USER NOTIFY가 그의 부모 바인딩 유닛인 SM이 바운딩 된 후에 동적으로 결합 될 수 있다는 것은 명백하다. 우리는 이 정보를 기반으로 적절한 동적 바인딩 테크닉을 탐구할 수 있었다.

반면 재구성 과정에서 예외 처리 전략이나 다양한 명세(예: 행위, DFD, 재구성명세)들간의 일관성 분석을 위한 정형적 기반탐구와 같은 동적 재구성의 다른 문제점들을 고려하기 위해서는 본 방법을 확장해야 한다.

## 6. 관련 연구

프로덕트 라인 엔지니어링에서 대부분의 노력(변경점의 발견과 명세, 일관성이 있는 관리, 제품 코드 생성을 위한 테크닉[1, 2, 3, 4])들은 정적 구성을 가진 제품들의 변경점을 가진 핵심 자산들을 개발하는데 기울여져왔다. 최근에 제안된 Reconfigurable Product Line UML Based SE Environment(RPLUSEE)[5]에서는 소프트웨어 동적 재구성의 패턴을 제안하였다. 여기서는 동적 재구성 정보가 저장되는 위치에 따라 이러한 패턴들을 마스터-슬레이브, 중앙집중방식, 클라이언트-서버, 탈중앙화방식으로 나눈다. 이 방법은 또한 재구성 스테이트 차트와 재구성 트랜잭션 모델을 제공하며, 동적 재구성 유닛의 하이레벨 명세에 초점을 맞추고 있다. 하지만 재구성 컴포넌트의 발견, 디자인, 구현에 대해서 깊이 있는 테크닉이나 가이드라인을 제시하지 못하고 있다. 한편 동적 재구성에 대한 대부분의 연구(예: 컨텍스트 인식 및 자기치유 시스템)는 각 분야에서 특정한 문제들(수행 상황의 동적 변화를 수용하기 위한 행위 모델[6], 소프트웨어나 하드웨어로부터 얻은 데이터를 분석을 통한 현재 상황 인식[10,11,12], 제품 구성과 소프트웨어 컴포넌트 버전의 자동 관리[7,8,9,13,14]) 에 초점을 맞추고 있다. 이러한 테크닉들은 동적 재구성에 있어서 필수적이지만 이러한 테크닉들을 어떻게 결합하여 제품마다 달라질 수 있는 구성의 정적/동적 변경점에 적용할 것인가에 대해서는 연구되지 않았다.

## 7. 결 론

프로덕트 휘처의 수가 증가하면서 휘처들과 그 변경점을 관리하는 것은 프로덕트 자산 개발자에게 커다란 부담이 되고 있다. 이 어려움은 특히 동적 재구성이 가능한 핵심 자산을 개발하는

데 있어서 더욱더 가중된다. 우리는 휘처들을 같은 바인딩 시점을 가지는 그룹인 휘처 바인딩 유닛이라는 단위로 그룹화하고 이 휘처 바인딩 유닛을 주요 디자인 드라이버로 하여 이러한 어려움들을 완화시켰다. 우리는 동적 재구성에 있어서 중요한 요소인 제품 변경점 관리의 용이성과 재구성의 변화의 가시성을 향상하기 위하여 노력하였다.

우리는 재구성기의 개발을 위해서 동적 재구성기의 세가지 관점을 제안하였고 각각의 관점을 명세하는 방법을 제시하였다. 제시된 단계별 엔지니어링 가이드라인을 통해서 프로덕트 라인의 복잡한 컨텍스트를 분석하고 정의할 수 있었고 컨텍스트 정보를 재구성 정보와 연결 시킬 수 있었다.

현재는 바인딩 유닛의 행위와 동적 재구성 명세의 일관성을 분석하기 위해서 정형 프레임워크를 개발하고 있다. 이후에는 동적 재구성의 의존성을 높이기 위한 예외 처리 전략에 대한 연구를 할 계획이다.

## 참고문헌


[1] P. Clements and L. Northrop, Software Product Lines: Practices and Pattern, Addison Wesley, Upper Saddle River, NJ (2002)

[2] D.M. Weiss and C.T.R Lai, Software Product-Line Engineering: A Familly-Based Software Development Process, Reading, MA: Addison Wesley Longman, Inc., (1999)

[3] K. Kang, J. Lee, and P. Donohoe, Feature-Oriented Product Line Engineering, IEEE Software, 19(4), July/August (2002) 58-65

[4] J. Bosch, G. Florijin, D. Greefhorst, J. Kuusela, J.H. Obbink, K. Pohl, Variability Issues in Software Product Lines, In: van der Linden, F. (eds.): Software Product Family Engineering. Lecture Notes in Computer Science, Vol. 2290. Springer-Verlag, Berlin Heidelberg (2002) 13-21

[5] H. Gomaa and M. Hussein, Dynamic Software Recon-figuration in Software Prodict Families, In: van der Linden, F. (eds.): Software Product Family Engineering, Lecture Notes in Computer Science, Vol.3014 Springer-Verlag, Berlin Heidelberg (2004) 435-444

[6] J. Kramer and J. Magee, The Evolving Philosophers Problem: Dynamic Change Management, Transaction on Software Engineering, Vol. 16, No. 11, November (1990) 1293-1306

[7] D. Garlan and B. Schmerl, Model-based Adaptation for Self-Healing Systems, Proceeding of the Workshop on Self-Healing Systems (WOSS'02), Nov. 18-19 (2002) 27-32

[8] A.G. Ganek and T.A. Corbi, The drawing of the autonomic computing era, IBM Systems Journal, Vol. 42, No. 1 (2003) 5-18

[9] P. Oreizy, et al., An Architecture-Based Approach to Self-Adaptative Software, IEEE Intelligent Systems, May/June (1999), 54-62

[10] S.S. Yau, F. Karim, Y. Wang, B. Wang, and S.K.S. Gupta, Reconfigurable Context-

Sensitive Middleware for Pervasive Computing, Pervasive Computing, July/ September (2002) 33-40

[11] B. Schilit, N. Adams, and R. Want, Context-Aware Computing Applications, Proceedings of IEEE Workshop Mobile Computing Systems and Applications, IEEE CS Press, Los Alamitos, Calf. (1994) 85-90

[12] M. Mikic-Rakic and N. Medvidovic, Architecture-Level Support for Software Component Deployment in Resource Constrained Environments, Proceedings of First International IFIP/ACM Working Conference on Component Deployjment, Berlin, Germany, (2002) 31-50

[13] A. van der Hoek, A.L. Wolf, Software release management for component-based software, Software Practice and Experience, Vol.33 (2003) 77-98

[14] R.S. Hall, D.M. Heimbigner, and A.L. Wolf, A cooperative approach to support software deployment using the software dock, Proceedings of the 1999 International Conference on Software Engineering, ACM Press: New York, NY (1999) 174-183

[15] J.P. Sousa and D. Garlan, Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments, Proceeding of the 3rd Working IEEE/IFIP Conference on Software Architecture, Kluwer Academic Publishers (2002) 29-4317

[16] G. Banavar and A. Bernstein, Software infrastructure and design challenges for ubiquitous computing applications, Communications of ACM, Vol.45, Issue 12 (2002) 92-96

[17] M. Kim, J. Lee, K. Kang, Y. Hong, and S. Bang, Reengineering Software Architecture of Home Service Robots: A Case Study, Proceedings of the 27th International Conference on Software Engineering, ACM Press: New York, NY (2005) 505-512

[18] J. Lee and K. Kang, Feature Binding Analysis for Product Line Component Development. In: van der Linden, F. (eds.): Software Product Family Engineering. Lecture Notes in Computer Science, Vol. 3014. Springer-Verlag, Berling Heidelberg (2004) 266-276

[19] N. Medvidovic, D.S. Rosenblum, R.N. Taylor, A Language and Environment for Architecture-Based Software Development and Evolution. Proceedings of the 21st International Conference on Software Engineering, ACM Press: New York, NY (1999) 44-53

[20] N. Medvidovic and R.N. Taylor, Exploiting architect-tural style to develop a family of applications, Software Engineering, IEE Proceeding, Vol. 144, No 5-6. October /December (1997)

[21] K. Czarnecki, U. Eisenecker. Generative Program-ming: Methods, Tools, and Applications, Reading, MA: Addison Wesley Longman, Inc. (2000)

[22] J. Kramer et al., Software Architecture

Description, Software Architecture for Product Families: Principles and Practice, M. Jazayeri et al., ed., pp. 31-64. Upper Saddle River, N.J.: Addison-Wesley (2000)

[23] R. Roshandel, A. van der Hoek, M. Mikic-Rakic, N. Medvidovic, Mae ? A System Model and Environment for Managing Architectural Evolution, ACM Transactions on Software Engineering and Methodology, 13(2) (2004) 240-276

[24] K. Kim, Dynamic Reconfiguration of Architecture based on Component Interactions, Master Thesis, Dept. of CSE, POSTECH. (2006)

## 저자약력

**이 재 준**

1991년 서강대학교 수학과(이학사)
1998년 포항공과대학교 정보통신학과(공학석사)
2006년 포항공과대학교 컴퓨터공학과(공학박사)
1993년~2000년 LG 정보통신 중앙연구원 주임 연구원
2000년~2001년 포항공과대학교 정보통신 연구소 전임 연구원
현재 독일 Fraunhofer IESE 연구원
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
이 메 일 : jaejoon.lee@iese.fraunhofer.de

**강 교 철**

1973년 고려대학교 계산통계학과 (이학사)
1976년 University of Colorado (공학석사)
1982년 University of Michigan (공학박사)
1982년~1984년 University of Michigan, Visiting Professor
1984년~1985년 Bell Communications Research, Technical Staff
1985년~1987년 AT&T Bell Labs., Technical Staff
1987년~1992년 SEI, Carnegie Mellon University, Senior Member, Project Leader
1992년- 현재 포항공과대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처, 소프트웨어 명세
이 메 일 : kck@postech.ac.kr

**김 경 석**

2004년 숭실대학교 컴퓨터학부(공학사)
2006년 포항공과대학교 컴퓨터공학과(공학석사)
현재 삼성전자 기술총괄 CTO전략실 개발혁신팀 재직
관심분야 : 소프트웨어 제품 라인 공학, 소프트웨어 재사용, 소프트웨어 아키텍처
이 메 일 : ks0927.kim@samsung.com

**고 재 훈**

2006 포항공과대학교 컴퓨터공학과(공학사)
현재 포항공과대학교 소프트웨어 연구실 소속 통합 과정
관심분야 : 소프트웨어 제품 라인 공학, 서비스 오리엔티드 아키텍처
이 메 일 : roengram@postech.ac.kr