TToSA : 제품계열공학으로의 전이를 위한 임베디드 소프트웨어의 모델 기반 아키텍처 변환기

홍 장 의[†] · 오 기 영[††] · 김 종 필[††]

요 약

임베디드 소프트웨어의 응용범위 확대, 요구기능의 복잡성 증가, 그리고 제품개발의 신속성이 요구됨에 따라 소프트웨어 개발업체에서는 기존의 개발 환경에서 진화하여 제품계열공학에 근거한 소프트웨어 생산 환경으로의 전환을 시도하고 있다. 이를 위해서는 대상 시스템에 대한 소프트웨어 아키텍처의 개발이 필수적으로 요구되는데, 본 연구에서는 개발업체가 기존의 소프트웨어 자산으로 보유하고 있는 구조적 모델을 소프트웨어 아키텍처 모델로 변환하기 위해 요구되는 방법 및 지원 도구를 제안한다. 제안하는 아키텍처 변환기는 기존 임베디드 소프트웨어 개발 환경에 대한 큰 변화 없이 제품계열공학의 소프트웨어 개발환경으로 접근할 수 있도록 지원한다.

키워드 : 임베디드 소프트웨어, 구조적 모델링, 소프트웨어 아키텍처

TToSA: An Architecture Model Translator toward Embedded Software Product Line Engineering

Jang-Eui Hong[†] · Gi-Young Oh[††] · Jong-Phil Kim[††]

ABSTRACT

Along with the enlargement of application scope, the growth of requirements complexity, and the fast development of product for embedded system, lots of industries developing embedded software try to evolve their traditional development environment into the new paradigm such as product line engineering approach. In order to sufficiently support the evolution, software architecture is essentially required to develop the embedded software. In this paper, we propose a tool, named TToSA which translates the conventional software models to software architecture models. Our TToSA is developed with the critical implication about that an industry can approach toward the new development paradigm without the big change of the existing software development method.

Key Words : Embedded Software, Structured Modeling, Software Architecture

1. 서 론

기존의 임베디드 소프트웨어 개발에 있어서는 구조적 분석 및 설계 방법에 의한 소프트웨어 개발이 대부분을 차지하고 있다. 이러한 현상은 내장되는 소프트웨어의 소스 코드가 대부분 C 언어를 이용해 생성되기 때문이다. 구조적 분석 및 설계 방법론에 의한 소프트웨어 개발에 익숙한 기존의 개발 조직에서는 분석 및 설계 과정에서 다양한 산출물을 작성해 왔으며, 이러한 산출물들은 임베디드 소프트웨어 개발 환경에서의 핵심 자산으로 여겨지고 있다[1, 2].

그런데 구조적 방법론을 적용하여 소프트웨어를 개발해 오던 조직에서 급변하는 사용자의 변화에 대응하고 시장 점유의 확대를 위하여 신속하고 체계적인 소프트웨어 개발 방법을 도입하려는 시도가 생겨나고 있다. 아키텍처 기반의 소프트웨어 개발이 이러한 조직의 요구에 적합한 기술로 고려되고 있으나 기존에 보유하고 있던 유용한 소프트웨어 자산을 무시할 수 없는 입장에 놓여있다.

따라서 본 연구에서는 기존에 구조적 방법론에 의하여 개발된 산출물을 이용하여 소프트웨어 아키텍처를 생성하기 위한 방법을 연구하고, 이를 위한 모델 변환기를 개발하였다. 구조적 방법론의 주요 산출물인 태스크 모델과 관련 정보를 입력 받아 소프트웨어 아키텍처로 변환하고, 변환된 아키텍처에 대한 선언적 명세언어(아키텍처 기술언어)를 이용하여 코드를 생성하도록 하였다. 이러한 변환기의 개발은 소프트웨어 개발 방법에 대한 변화를 시도하는 조직에서 (1) 소프트웨어 아키텍처를 보다 쉽고 빠르게 생성할 수 있도록 지원할 뿐만 아니라 (2) 아키텍처 기술 언어를 기반으

※ 본 연구는 2005학년도 충북대학교 학술 연구지원 사업의 연구비 지원에 의해 수행되었음.
† 종신회원 : 충북대학교 컴퓨터공학 조교수
†† 준 회 원 : 충북대학교 전자계산학과 석사과정
논문접수 : 2006년 5월 30일, 심사완료 : 2006년 9월 11일

로 하는 재사용 가능한 소프트웨어 컴포넌트로의 래핑(wrapping)을 손쉽게 지원할 수 있는 장점을 제공한다.

본 논문의 구성은 2장에서는 태스크 모델과 소프트웨어 아키텍처의 구성요소들에 대하여 살펴보고, 3장에서는 모델 변환을 위한 규칙들의 정의와 아키텍처 생성절차를 기술하였다. 4장에서는 제안하는 변환기의 설계 및 개발 내용을 제시하고, 5장에서는 예제 시스템에 대한 적용 사례를, 6장에서는 기존의 연구들과 비교 분석을, 그리고 결론을 7장에 기술하였다.

2. 태스크 모델과 소프트웨어 아키텍처

2.1 태스크 모델

태스크 모델은 구조적 방법론의 기본 설계 단계의 산출물이다. 태스크 모델을 정의하면 다음과 같다.

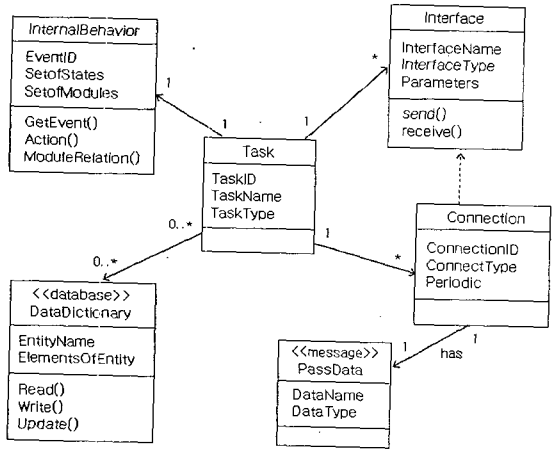
[정의 1] 구조적 모델인 태스크 모델 M_t는 다음과 같이 6개의 요소를 갖는 터플(tuple)로 정의한다.

$$M_t = <T, C, D, S, F, B>$$

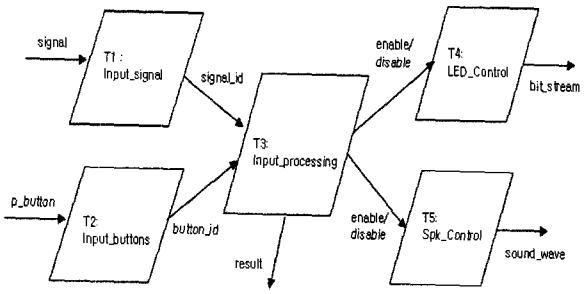
이 정의로부터 태스크 모델을 구성하는 각 요소들은 다음과 같이 정의한다.

- T = 태스크 t_i의 집합
- C = 태스크 t_i의 연결자 $c_{ti,i}$ 들의 집합
- D = 연결자 $c_{ti,i}$에 대해 정의된 데이터
- S = 비 휘발성 데이터 집합
- F = 태스크 t_i가 갖는 인터페이스 $F_{ti,i}$
- B = 태스크 t_i의 내부 행위 ■

[정의 1]에서 제시된 태스크 모델의 구성요소들을 기반으로 구성요소들의 속성과 그들간의 관계는 (그림 1)과 같은 메타 모델에 의해 정의될 수 있다. 메타 모델에서의 클래스들에 대하여 설명하면 다음과 같다.



(그림 1) 태스크 모델의 정적구조



(그림 2) 통신 단말 입력부의 태스크 모델 예시

- 클래스 "Connection"은 태스크들간의 연결을 표현하며, 이는 단방향 또는 양방향 등과 같은 타입을 갖는다.
- 클래스 "Interface"는 태스크가 어떠한 인터페이스를 갖고 연결되는 가를 정의한다. 동기, 비동기 등의 타입 정보를 포함한다.
- 클래스 "InternalBehavior"는 태스크의 내부 행위를 묘사하는 모듈로 정의된다.
- 스테레오타입 <<database>>로 선언된 클래스는 태스크 모델이 접근하는 공유 데이터에 대한 정의이며, <<message>> 타입의 클래스는 연결자에 부여되는 파라메타에 대한 표현이다.

(그림 2)에 주어진 간단한 태스크 모델의 예는 이동통신 단말 장치의 입력 처리 부분에 대한 간략한 태스크 모델을 표현한다. 신호의 수신과 사용자에 의한 버튼의 입력을 받아 수신 데이터를 처리하고 처리 결과에 따라 필요한 동작을 수행하는 과정을 보여준다.

2.2 소프트웨어 아키텍처 정의

제품 계열 공학에 근간한 소프트웨어를 개발하기 위해서는 기본적으로 (1) 소프트웨어 아키텍처, (2) 컴포넌트 및 컴포넌트 구현, (3) 아키텍처 기반 개발 프로세스가 정의되어야 한다[3, 4, 19]. 이 중에서도 소프트웨어 아키텍처는 제품계열 공학의 기본 축을 이루고 있다[5-7, 15]. 본 절에서는 D. Garlan이 제시한 소프트웨어 아키텍처의 구성 요소를 중심으로 정의하였다[8, 9].

[정의 2] 소프트웨어 아키텍처 M_{sa}는 다음과 같은 7개의 요소에 의해 정의한다.

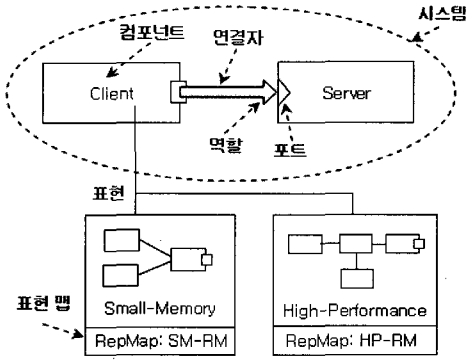
$$M_{sa} = <C_o, C_t, S_m, P_t, R_o, R_p, R_m>$$

이로부터, C_o는 컴포넌트, C_t는 연결자, S_m은 시스템, P_t는 포트, R_o는 역할, R_p는 표현(representations), R_m은 표현 맵(rep-map)을 의미한다. ■

[정의 2]에 기술된 소프트웨어 아키텍처의 구성요소들에 대하여 도식화하면 (그림 3)과 같다.

〈표 1〉 아키텍처 구성요소들의 예제들

구성요소	예 제
컴포넌트	Clients, Servers, Filters, Objects, Device drivers, Database 등
연결자	Pipe, Procedure call, Event broadcast, Message passing 등
시스템	Client/Server, Web, Host-terminal 등
포트	Procedure signature, Method interface
역할	caller/callee, sender/receiver 등



(그림 3) 소프트웨어 아키텍처 구성요소

(그림 3)의 아키텍처 구성요소로부터 컴포넌트는 시스템을 구성하는 기본 단위이며, 이들은 연결자에 의해 상호작용이 표현된다. 컴포넌트의 인터페이스는 포트에 의해 정의된다. 포트는 컴포넌트와 컴포넌트, 컴포넌트와 환경간의 상호 작용에 대한 정점(point)으로 표현되며, 한 컴포넌트는 서로 다른 여러 타입의 포트를 가질 수 있다. 연결자는 역할에 의해 정의되는 인터페이스를 갖는다. 포트가 컴포넌트 인터페이스 부분이면, 역할은 연결자의 인터페이스라고 볼 수 있다[9].

<표 1>은 소프트웨어 아키텍처를 구성하는 구성요소들에 대한 대표적인 예들을 정리한 것이다. 이러한 예를 통해 아키텍처 구성요소들의 의미를 보다 정확히 파악할 수 있다.

아키텍처의 정의에 있어서 표현이라 함은 컴포넌트에 대한 내부 행위에 대한 표현이며, 표현 맵이라 함은 임의의 컴포넌트에 대하여 표현 가능한 다양한 행위 모델이 존재하고, 이는 시스템 요구사항에 따라 서로 다른 모델로 매핑(구현)되어질 수 있음을 의미한다.

3. 모델 변화 규칙

본 절에서는 구조적 모델의 산출물인 태스크 모델을 소프트웨어 아키텍처로 변환하기 위한 규칙을 정의한다. 이러한 규칙의 정의는 제품 계열 공학에 의한 임베디드 소프트웨어 생성을 위하여 기존의 설계 자산(assets)을 재사용하기 위함이다. 앞의 2장에서 정의된 태스크 모델의 구성 요소와 소프트웨어 아키텍처 구성 요소간의 변환 규칙은 다음과 같다.

3.1 변환 시스템의 정의

모델의 구성요소에 대한 변환을 위해 다음과 같은 변환 시스템을 정의한다.

[정의 3] 태스크 모델에 대한 아키텍처로의 변환을 위한 변환 시스템 S는 다음과 같이 정의한다.

$$S = \langle M_t,\ b,\ a,\ x:=e;\ M_{sa} \rangle$$

여기서 M_t와 M_{sa}는 각각 소스 모델과 타겟 모델이며, b는 불리언 값, a는 변환을 위한 액션, 그리고 $x:=e$는 원소의 배정을 의미한다. 즉, 소스 모델에 특정 요소가 존재한다면, a의 실행에 의해 타겟 모델의 요소로 배정된다. ■

[정의 3]에서 변환을 위한 동작(액션)은 정형화된 변환 규칙을 기준으로 이루어지며, 변환 규칙에 대한 정의는 다음과 같다.

[정의 4] 변환 시스템을 위한 모델 요소의 변환 규칙은 다음과 같은 패턴을 갖는다.

$$\frac{X \qquad P_1, \ldots, P_n}{Y \qquad Q_1 \ \cdots \ Q_n}$$

여기서 X는 소스 모델을 구성하는 요소의 인스턴스이며, P_i는 인스턴스 X에 대한 속성을 정의한다. Y는 타겟 모델에서 X에 대응하는 요소의 변환된 인스턴스이며, Q_i는 P_i를 입력으로 하는 Y의 속성 결정 함수이다. ■

[정의 4]에 의거하여 태스크 모델의 각 구성요소들에 대하여 적용된 변환 규칙은 다음과 같다. 여기서 I_x는 모델 구성요소 X의 인스턴스를 의미하며, 함수 f는 변환시스템의 $x:=e$를 실행하는 함수이다.

3.1.1 컴포넌트 변환 규칙

$$\frac{I_t \in \{T \cup S\} \quad \exists (id,\ type) \text{ for } I_t}{I_{Co} \in C_o \quad f(id(I_t)) = id(C_o) \quad f(type(I_t)) = type(C_o)}$$

위의 변환 규칙은 태스크 T와 데이터의 집합 S는 모두 아키텍처 모델이 컴포넌트로 매핑된다. 이때 태스크에 대응되어 있는 속성들은 해당 컴포넌트의 요소를 정의하는데 사용됨을 나타낸다.

다른 요소들에 대한 선언 속성들도 이와 유사한 변환 규칙을 따른다.

3.1.2 연결자 변환 규칙

$$I_c \in C \quad \forall I_c \in \{ \bullet T_i \cup T_i \bullet \} \wedge$$
$$\exists (D_{Tt}, type) \text{ for } I_c$$
$$I_{Ct} \in C_t \quad f(\bullet T_i) = type_{IN}(P_t),$$
$$f(T_i \bullet) = type_{OUT}(P_t)$$
$$f(type(I_c)) = protocol(C_t) \wedge I_{Ro}$$
$$f(D(I_c)) = name(P_t)$$

태스크 모델의 연결자는 특정 태스크의 입력과 출력 연결자로 구분되는데, 이들은 포트의 타입을 결정하게 되며, 연결자의 타입 즉, 화살표의 방향과 비동기/동기 특성은 아키텍처 모델에서 프로토콜과 역할을 정의하기 위해 사용된다. 또한 태스크 모델에서 연결자에 부여되는 데이터의 집합은 컴포넌트 포트의 이름으로 매핑된다.

3.1.3 인터페이스 변환규칙

$$I_F \in F \quad \exists I_F \in \{T_i \times T_j\} \wedge (I_F \rightarrow \forall I_c)$$
$$I_{Ct} \in C_t \quad f(I_F) = Attachment(C_o \times C_t)$$

태스크 모델의 인터페이스 정보는 모든 태스크들에 대하여 존재하며, 인터페이스가 존재한다면 연결자도 존재하게 된다. 태스크간 인터페이스는 컴포넌트가 갖는 포트간의 연결점(attachment)을 정의하기 위해 사용된다.

3.1.4 내부 행위 변환규칙

$$I_B \in B \quad \exists I_B \in \{B_i \mid i = 1, ..,n\}$$
$$I_{Rp} \in R_m \quad f(I_B) = \mathrm{Rp} \ltimes C_{Rm}$$

태스크의 내부 행위는 다양한 형태(n개의 형태)로 표현될 수 있으며, 이는 컴포넌트의 구현은 부여된 제약사항, C_{Rm}에 따라 컴포넌트의 내부 구조로 매핑된다.

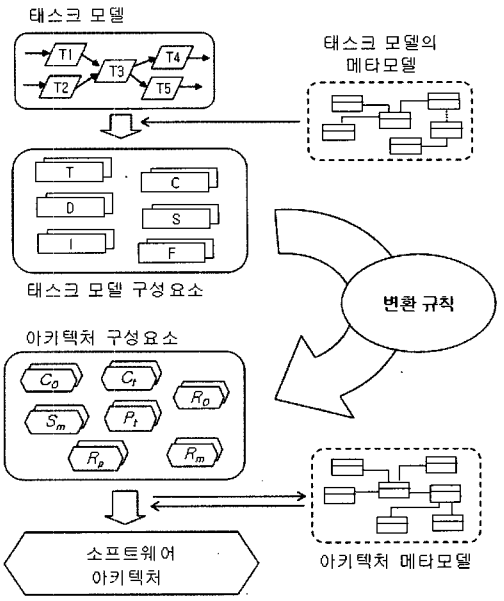
태스크 모델을 구성하는 전체 요소들은 시스템 구현의 형상을 나타낸다. 이는 아키텍처 모델의 시스템 S_m에 대응하는 소프트웨어 구조로 매핑된다.

3.2 아키텍처 생성 절차

앞서 정의된 규칙을 기준으로 소프트웨어 아키텍처를 생성하기 위한 절차는 (그림 4)와 같이 정의할 수 있다.

(그림 4)에서 보는 바와 같이 비주얼한 태스크 모델은 메타 모델의 형태로 저장되며, 이는 각각 태스크의 구성요소들로 분해된다.

분해된 태스크 모델의 요소들은 3.1절에서 제시한 변환 규칙에 따라 각각 대응하는 소프트웨어 아키텍처의 컴포넌트들로 전환되고, 이들 정보는 아키텍처 모델의 메타 구조에 저장된다. 이들 저장된 정보를 이용하여 비주얼한 아키텍처 모델로 표현되거나, 동등한 정보의 기술언어(descriptive language)로 나타낸다.



(그림 4) 모델 변환의 기본 절차

4. 아키텍처 변환기

4.1 변환기 개발 요구사항

구조적 분석과정에 작성된 태스크 모델이 소프트웨어 아키텍처 모델로 전환되기 위해서는 태스크 모델의 메타 모델에 근거하여 XMI 타입의 파일로 저장되어야 한다[18]. XMI 타입의 파일을 읽어 들여 변환 규칙에 따라 아키텍처를 생성한다. 본 연구를 통해 개발한 TToSA (Translator of structured Task model to Software Architecture) 변환기 시스템의 핵심 요구사항을 정리하면 다음과 같다.

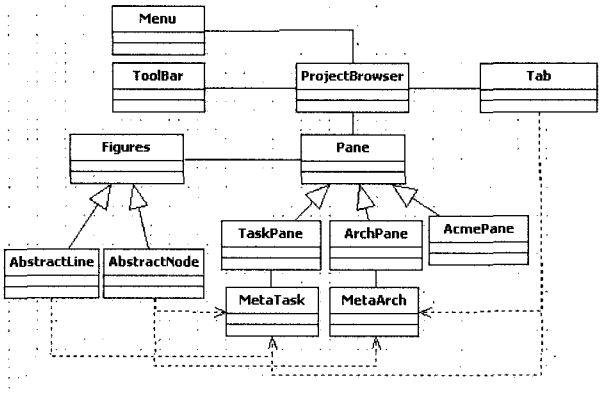
- 기존에 작성된 태스크 모델을 활용할 수 있어야 하며, 필요시 태스크 모델에 대한 수정도 가능하다.
- 임포트(import)된 태스크 모델에 대해서는 다시 XMI 타입의 파일로 저장 및 출력이 가능하다.
- 태스크 모델의 각 태스크들에 대하여 요구될 수 있는 상세 정보에 대해서는 추가 정의가 가능하다.
- 정제된 태스크 모델은 Garlan의 의해 정의된 비주얼 아키텍처 모델과 Acme 언어[9]로의 변환이 가능해야 한다.

4.2 TToSA 변환기의 설계

TToSA 변환기의 핵심적인 기능은 크게 다음의 4가지 기능으로 구분된다.

(1) 태스크 모델 편집기능 : 태스크 모델의 편집기는 XMI 타입으로 입력된 태스크 모델을 수정하거나 새롭게 작성하기 위해 지원되는 기능이다. 모델의 표현은 Gomma가 제안하는 CODARTS 방법에서의 태스크 모델을 따른다[2].

(2) XMI 임포트 및 익스포트 기능 : 기존의 레거시 시스템



(그림 5) TToSA 시스템의 클래스 다이어그램

에 대하여 작성된 모델이나 편집기능을 통해 작성된 태스크 모델을 XMI을 이용하여 관리하기 위한 기능이다.

(3) 아키텍처 변환기능 : 태스크 모델로부터 소프트웨어 아키텍처 모델을 생성하기 위한 기능이다. Garlan에 의해 제시된 비주얼 모델의 생성이 가능하다.

(4) Acme 생성기능 : 변환된 아키텍처의 비주얼 모델이 추상화되어 있기 때문에 소프트웨어 아키텍처에 대한 정보를 상세히 담고 있는 Acme 기술 언어로 작성하기 위한 기능이다.

이와 같은 주요 기능의 개발을 위해 설계된 TToSA 시스템의 최상위 클래스 다이어그램은 (그림 5)와 같다. (그림 5)의 클래스 다이어그램에서 추상화 클래스 "Figures"는 태스크와 아키텍처의 심볼들을 정의하는 클래스이며, 추상화 클래스 "Pane"은 TToSA 도구의 작업패널을 정의한 클래스들이다.

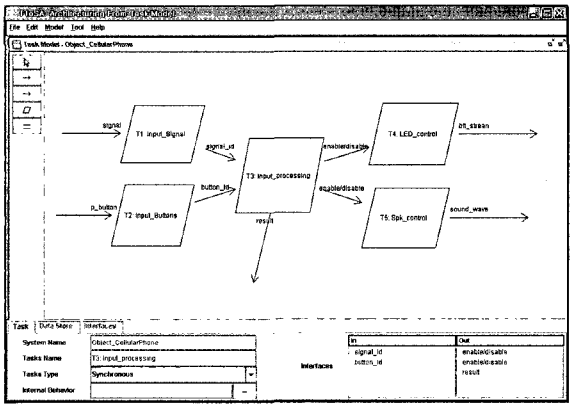
또한 클래스 "MetaTask"와 "MetaArch"는 각각 태스크 모델과 아키텍처 모델의 정보를 저장하는 메타 모델에 대응된다. TToSA 시스템은 이클립스 프레임워크(eclipse 3.2)상에서 개발되었다.

5. 사례 시스템 적용

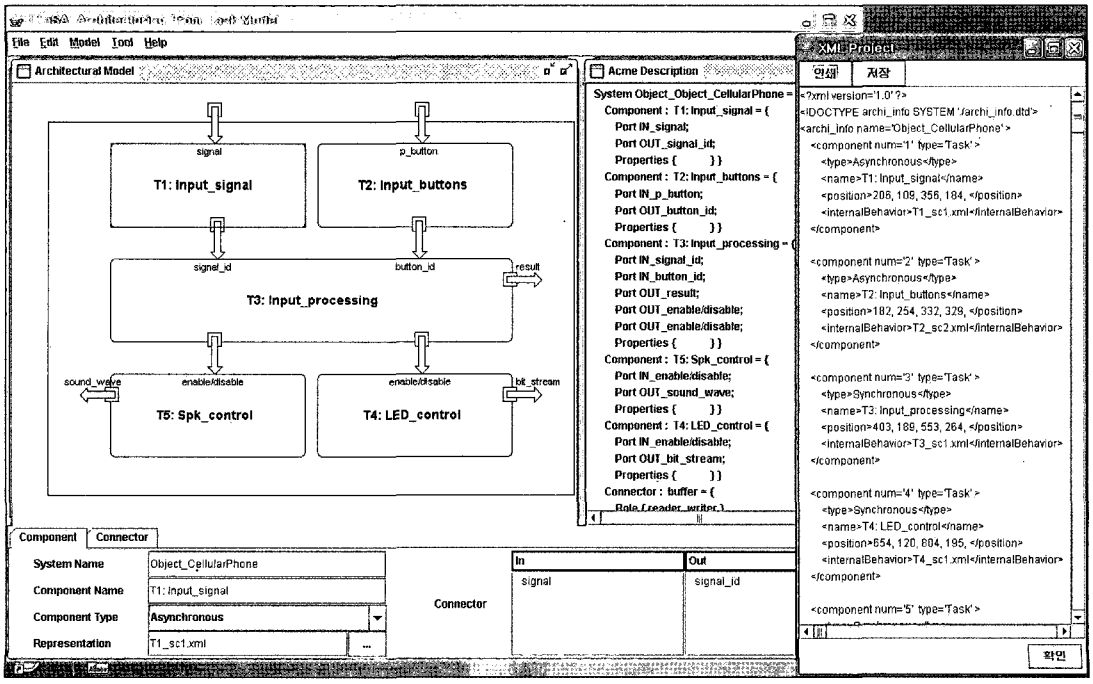
TToSA 변환기를 이용한 사례 시스템의 적용을 설명하기 위해 (그림 2)에서 사용되었던 이동통신 단말장치의 입력처리부에 대한 태스크 모델을 이용한다. (그림 2)에서 보는 바와 같이 이 태스크 모델은 크게 5개의 태스크를 갖는다.

5.1 태스크 모델 정의

TToSA 변환기를 이용한 태스크 모델의 사용자 인터페이스는 (그림 6)과 같다.



(그림 6) TToSA의 태스크 모델 편집기



(그림 7) 변환된 소프트웨어 아키텍처 모델

(그림 6)에서 하단의 작업영역은 Task, Data Store, Interface의 세부 속성을 정의하기 위한 필드로 구성되어 있으며, 이들은 사용자에 의해 수정되거나 보완될 수 있다.

5.2 아키텍처 모델로의 변환

TToSA 도구의 Tool 메뉴는 태스크 모델을 아키텍처로 변화하고, 또한 대응되는 Acme 언어를 생성하기 위한 기능을 제공한다. (그림 7)은 (그림 6)의 태스크 모델을 입력으로 변환된 아키텍처 모델이다.

5.3 Acme 기술언어 생성

변환된 소프트웨어 아키텍처에 대하여 Acme 언어로의 자동 생성이 가능하다. Acme 언어는 주어진 시스템에 대하여 컴포넌트, 포트, 역할, 컴포넌트 속성(proper- ties), 그리고 포트와 연결자의 역할을 매핑하는 Attachment로 정의한다. 주어진 예제 시스템에 대하여 생성된 Acme 언어로의 기술에서 (그림 7)의 오른쪽 하단에 숨겨진 부분에 대한 정의는 (그림 8)과 같다.

(그림 8)의 Acme 정의를 살펴보면, 컴포넌트 T1부터 T5들은 각각 입력 정보를 갖는 포트와 출력정보를 갖는 포트로 정의되었다. 동일한 포트이지만 타입의 구분하여 정의할 수 있다. 연결자는 buffer와 eventCast 타입이 존재하는데, buffer 타입의 경우는 reader/writer의 역할을 갖고, Event-Cast의 경우는 sender/receiver 역할을 갖게 된다. Attachment 필드에서는 연결자에 의한 컴포넌트간 연결 정보가 정의된다.

```
System Object_CellularPhone = {
        :
    Connector :  buffer = {
        Role { reader, writer }
        Properties { synch : boolean = True;
                     max-roles : integer = 2;
                     protocol : FIFO } }
    Connector :  eventCast = {
        Role { sender, receiver }
        Properties { synch : boolean = True;
                     max-roles : integer = 2;
                     protocol : null } }

        :

    Attachment {
        T1: Input_Signal.signal to ExternalInput.receiver;
        T1: Input_Signal.signal_id to eventCast.sender;
        T2: Input_Buttons.p_button to ExternalInput.receiver;
        T2: Input_Buttons.button_id to eventCast.sender;
        T3: Input_processing.signal_id to eventCast.receiver;
        T3: Input_processing.button_id to eventCast.receiver;

        :

        T5: Spk_control.enable/disable to buffer.reader;
        T5: Spk_control.sound_wave to ExternalOutput.sender;
    }
}
```

(그림 8) 변환된 아키텍처의 Acme 기술

5.4 제품계열공학 환경으로의 접근

제품계열공학에 근간한 소프트웨어를 개발하기 위해서는 기본적으로 (1) 소프트웨어 아키텍처, (2) 컴포넌트 및 컴포넌트 구현, 그리고 (3) 개발 프로세스가 정의되어야 한다. TToSA 변환기를 이용한 제품계열공학 방식으로의 전환은 기존의 소프트웨어 자산을 재사용 가능한 컴포넌트로 활용하고, 이를 기반으로 소프트웨어 아키텍처를 생성하기 때문에 보다 쉽게 제품계열공학으로의 접근이 가능하다고 할 수 있다.

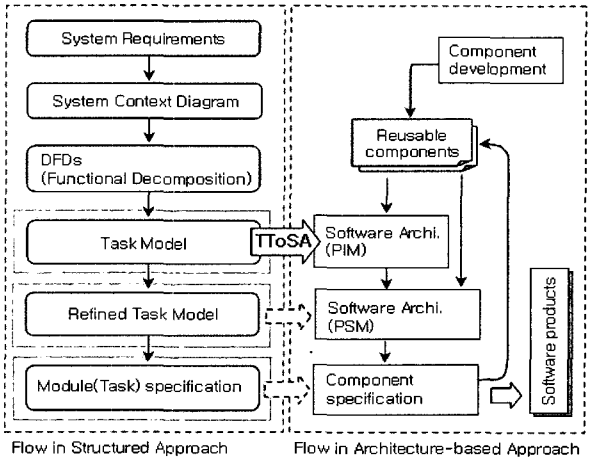
구조적 방법론을 근간으로 하는 소프트웨어 개발 프로세스를 아키텍처 기반의 개발 방식으로 전환하기 위해서는 (그림 9)에서 보는 바와 같이 프로세스의 연계가 필요하다.

기존에 구조적 방법론에 의해 개발된 산출물은 플랫폼 독립적인 모델과 의존적인 모델들로 매핑되어 전환하고, 신규 개발 모듈은 컴포넌트 타입으로 개발되어 재사용을 지원하게 된다.

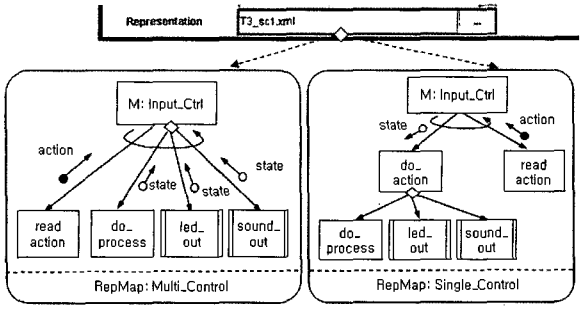
제품계열공학을 지원하기 위해서는 모델의 구성요소에 대한 가변성(Variability) 메커니즘도 지원하여야 한다. 즉, 모델의 구성요소가 목표로 하는 시스템의 특성 및 요구사항을 충족할 수 있도록 지원해야 한다. TToSA 시스템에서는 아키텍처를 구성하는 컴포넌트의 내부 행위를 바꾸기 위해 행위 모델에 대한 바인딩을 선택적으로 수행할 수 있도록 지원하였다.

(그림 10)에서는 이러한 변형의 개념을 지원하기 위한 TToSA 시스템의 지원 부분을 나타내고 있다. 도구의 왼쪽 하단에 존재하는 "Representation" 필드는 내부 행위 모델을 선택하기 위한 부분으로써, 구조차트 또는 상태머신으로 작성된 하위 모델을 포함하도록 한다.

제품계열공학에서 가변성 메커니즘은 아키텍처 레벨에서 아키텍처 구성요소에 대한 포함과 배제, 중복 배치, 그리고 버전 선택 등으로 지원될 수 있으며[6], 아키텍처를 구성하는 요소 레벨에서는 상속 및 재정의, 확장, 파라메터화 등에 의해 지원된다[6, 20, 21]. 가변성의 지원은 아키텍처 모델에서의 변경 점(variation point)을 관리하는 것이 필요한데, 이



(그림 9) 아키텍처 기반 하이브리드 프로세스



(그림 10) TToSA의 변형 지원 개념

는 제품계열 생산을 위한 변경의 수준을 정의하고, 변경 부분을 선택, 추적, 관리하기 위해 사용된다[22-25].

본 논문에서는 제품계열공학 환경으로의 전이를 지원하기 위하여 (그림 10)에서 보인 바와 같이 아키텍처를 구성하는 컴포넌트의 내부 행위에 대한 대체 개념으로 모델의 가변성을 제한하였다. 특히 임베디드 소프트웨어를 상품의 모델별로 생산하기 위해서는 아키텍처를 구성하는 컴포넌트가 디바이스 드라이버나, 특정 처리 모듈 등과 같은 모듈 단위로 이루어지며, 이에 대한 대체 개념을 통하여 제품의 모델 변경에 대한 소프트웨어 생산이 가능해질 수 있다고 할 수 있다.

6. 관련 연구 분석

임베디드 소프트웨어에 대한 아키텍처를 정의하기 위한 연구들은 상대적으로 많지 않다. Fricksk과 Kruchten은 UML 2.0을 이용하여 임베디드 소프트웨어의 아키텍처를 표현하고자 하였다[10, 11]. 이들의 연구에서 UML을 아키텍처 기술언어로 사용하고자 하였고, 이들은 UML이 제공하는 모듈화 및 재사용성 등과 같은 특성을 기반으로 제품 군 생산 방식에 용이한 소프트웨어 아키텍처를 정의하고자 하였다.

또 다른 연구들은 아키텍처 기술 언어들을 새롭게 정의하거나 확장하려는 연구들이다. 임베디드 시스템이 갖는 하드웨어적 특성과 시간 제약과 같은 속성을 정의하기 위하여 기술 언어의 신택스와 시맨틱을 확장하려는 시도이다[8, 12, 13]. 임베디드 소프트웨어의 아키텍처와 관련된 또 다른 연구 분야는 기존 레거시 소프트웨어 또는 소스 코드로부터 아키텍처를 복구(recovery)하기 위한 연구들인데[14-16], Vasconcelos는 시스템의 실행 과정에서 찾을 수 있는 동적 행위의 패턴을 중심으로 아키텍처 컴포넌트를 식별하였으며, Eixelsberger는 다양한 역공학 도구를 이용하여 소스코드로부터 아키텍처 속성 정보를 찾아내고, 이들을 합성함으로써 아키텍처를 정의하는 방법을 제시하였다. Ivkovic은 대규모 소프트웨어의 유지보수(Perfective maintenance)를 위해 도메인 기반의 아키텍처를 효과적으로 복구하기 위한 방법을 제시하였으며, 특히 아키텍처 적응성(adapta- bility)에 대한 대표적인 연구[17]인 Chung는 적응성 지식 베이스를 근간으로 하는 아키텍처를 생성하기 위한 프레임워크를 제안하였다.

그러나 이러한 연구들은 본 연구에서 제시하고자 하는 내용과는 다소 다른 관점에서의 접근들이다. 본 연구는 기존의 소프트웨어 개발 방법론으로부터 아키텍처 기반 방법론으로 전이하기 위한 순공학적 접근이라는 점과 코드 기반이 아닌 모델 기반의 접근이라는 측면에서 기존 연구와의 차별성이 있다고 판단된다.

7. 결 론

본 연구에서는 기존의 임베디드 소프트웨어 자산인 구조적 모델을 소프트웨어 아키텍처 모델로 변환하기 위한 지원 도구 TToSA를 개발하였다. 이는 구조적 접근 방법에 의한 소프트웨어 개발 환경으로부터 아키텍처 기반의 제품계열공학 환경으로의 접근을 시도하기 위한 첫 번째 단계이다. TToSA를 개발하기 위하여 먼저 구조적 방법론의 산출물에 대한 분석을 통해 아키텍처로의 변환 규칙을 정의하였으며, 또한 생성된 아키텍처에 대하여 Acme 기술 언어로의 매핑에 대하여 분석하였다. 소프트웨어 아키텍처를 기반으로 제품을 개발하는 것은 시장(market)의 요구에 대한 신속한 대처 및 제품의 계열 생산을 가능하게 하는 중요한 전략이기 때문에 제시한 연구 결과에 의한 임베디드 소프트웨어 개발 방식의 진화는 소프트웨어의 재사용성, 융통성, 적용성 등의 품질 향상에 기여할 수 있을 것으로 본다.

본 연구에 이은 향후의 연구는 생성된 아키텍처 모델을 기반으로 제품계열공학을 지원하기 위한 가변성 메커니즘에 대한 것이며, 이러한 연구를 통해 보다 효과적인 임베디드 소프트웨어 제품의 계열 생산이 가능해지리라 판단한다.

참 고 문 헌

[1] H. Gomaa, 'Software Design Methods for Concurrent and Real-Time Systems,' Addison-Wesley, 1993.

[2] Tom Demarco, 'Structured Analysis and System Specification,' Yourdon Press, 1979.

[3] H. Gomaa, 'Designing Software Product Lines with UML,' Addison-Wesley, 2005.

[4] C. Stoermer and M. Roeddiger, "Introducing Product Lines in Small Embedded Systems," LNCS Vol.2290, pp.101-112, 2001.

[5] P. Clements, R. Kazman, and M. Klein, 'Evaluating Software Architecture,' Addison-Wesley, 2002.

[6] L. Bass, P. Clements, and R. Kazman, 'Software Architecture in Practice,' Addison-Wesley, 2003.

[7] Junichi Miyao, "A Reliable Software Architecture for Complex Embedded Systems," Int'l Sym. on OORTDC, pp. 90-95, 1998.

[8] M. Shaw and D. Garlan, "Formulations and Formalisms in Software Architecture," LNCS Vol. 1000, Springer-Verlag, 1995.

[9] D. Garlan, R. Monroe, et al, "Acme: An Architecture

Description Interchange language," CASCON'97, Nov., 1977.
[10] G. Frick, B. Scherrer and K.D. Muller-Glaser, "Designing the Software Architecture of an Embedded System with UML 2.0," UML 2004 Workshop on Software Architecture Description and UML, pp.15, October, 2004.
[11] P. Kruchten, B. Selic, et al., "Describing Software Architecture with UML," ICSE 2001.
[12] Nenad Medvidovic and Richard N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE TSE, Vol.26, No.1, pp.70-93, Jan., 2000.
[13] N. Medvidovic, S. Malek, and M. M-Rakie, "Software Architecture and Embedded Software," Workshop of Software Eng. for Embedded Systems, Sep., 2003.
[14] A. Vasconcelos, and C. Werner, "Software Architecture Recovery based on Dynamic Analysis," Workshop on WMSWM, Oct., 2004.
[15] W. Eixelsberger, M. Ogris, et al, "Software Architecture Recovery of a Program Family," ICSE, 1988, pp.508-511
[16] I. Ivkovic, 'Enhancing Domain-Specific Software Architecture Recovery,' Thesis at Computer Science, University of Waterloo, 2002.
[17] L. Chung and N. Subramanian, "Adaptable architecture generation for embedded systems," Journal of Systems and Software, vol.71, pp.271-295, 2004.
[18] T.J. Grose, et al., 'Matering XMI,' Wiley Computer Publishing, 2002.
[19] 김행곤, 손이경, "프로덕트 라인 기반의 모바일 프로세스 개발 프로세스," 정보처리학회 논문지 Vol.12-D, No.3, pp.395-408, 06, 2005.
[20] D. Webber and H. Gomaa, "Modeling variability in Software product Lines with the variation point model," Science of Computer Programming, Vol.53, pp.305-331, 2004.
[21] I. Jacobson, et al, Software Reuse - Architecture Process and Organization for Business Success, ACM Press, 1997.
[22] M. Jaring and J. Bosch, "Representing Variability in Software Product Lines: A Case Study," SP&E, Vol.24, pp.69-100, 2004.
[23] K. Pohl and A. Metzger, "Variability Management in Software Product Line Engineering," ICSE06, pp.1049-1059, May, 2006.
[24] K. Berg, J. Bishop, and D. Muthig, "Tracing Software Product Line Variability," Proceedings of SAICSIT pp.111-120, 2005.
[25] K. Kang, Feature-Oriented Domain Analysis, TR CMU/SEI-90-TR-21, 1990.

홍 장 의

e-mail : jehong@chungbuk.ac.kr
1988년 충북대학교 전자계산학과(이학사)
1990년 중앙대학교 전자계산학과 (공학석사)
2001년 한국과학기술원 전산학(공학박사)
2002년 국방과학연구소 선임연구원
2002년~2004년 (주)솔루션링크 기술연구소장
2004년~현재 충북대학교 컴퓨터공학 조교수
관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 소프트웨어 프로세스

오 기 영

e-mail : ohgy@selab.chnugbuk.ac.kr
2006년 충북대학교 컴퓨터공학과(공학사)
2006년~현재 충북대학교 전자계산학과 석사과정
관심분야 : 임베디드 소프트웨어, 소프트웨어 아키텍처, 소프트웨어 품질공학

김 종 필

e-mail : kimjp@selab.chnugbuk.ac.kr
2006년 충북대학교 컴퓨터공학과(공학사)
2006년~현재 충북대학교 전자계산학과 석사과정
관심분야 : 임베디드 소프트웨어, 소프트웨어 아키텍처, 소프트웨어 품질공학