엔터프라이즈 프레임워크에서 닷넷 기반의 컴포넌트 워크플로우 가변성 구현 기법

Techniques of Component Workflow Variability Implementation based on .NET in the Enterprise Framework

노재우(Jae Woo Noh)*, 이승훈(Seung Hun Lee)**, 류성열(Sung Yul Rhew)***

초 록

컴포넌트 기반 개발(Component Based Development, CBD)은 미리 구현된 블록단위의 컴포넌트를 사용하여 소프트웨어 개발 비용 및 시간을 단축할 수 있다. 또한 컴포넌트는 내부의 상세 부분을 숨기고 인터페이스를 제공하여 대형 어플리케이션을 개발하는데 복잡성을 감소시킬 수 있다. 엔터프라이즈 프레임워크 환경에서의 패밀리(Family)의 공통적인 요구사항은 컴포넌트 내 업무 워크플로우 형태로 표현될 수 있으나, 다양한 패밀리 멤버(Family Member)의 요구사항에 특화된 워크플로우를 하나의 컴포넌트 내에 표현하는 것은 매우 어렵다. 따라서 패밀리 멤버(Family Member)를 위한 컴포넌트간 워크플로우 가변성 관리 및 구현 방법에 대한 필요성은 끊임없이 요구되어 왔다.

본 논문에서는 .NET 기반 하에 엔터프라이즈 프레임워크에서 컴포넌트 워크플로우 가변성을 위한 구현 기법을 제시한다. 제시된 컴포넌트 워크플로우 가변성 구현 기법은 엔터프라이즈 프레임워크에서 패밀리가 사용할 수 있는 공통 업무 워크플로우가 컴포넌트 내에 이미 생성되었음을 가정하며, 다양한 패밀리 멤버에 특화된 컴포넌트간의 워크플로우 구성 및 실행을 위한 기법으로 컴포넌트간 워크플로우의 확장성을 높여주게 된다.

ABSTRACT

CBD(Component Based Development) can make use of component of block unit that is implementation beforehand and shorten software development cost and time. Also, component can reduce complexity for the large application development because it can hide detail part of internal and offer interface. Common requirement of family in the enterprise framework environment can express workflow, but it is very difficult that express special workflow in single component on various family member's requirement. Therefore, necessity about workflow variability management between component for family member and implementation method had been required constantly.

This treatise presents implementation techniques for component workflow variability in enterprise framework under .NET base. Presented component workflow variability implementation techniques supposes that commonness business workflow that family can use in enterprise workflow was created already in component, and improve workflow's extensity between component by workflow composition between special component and techniques for practice to various family member.

키워드 : 엔터프라이즈 프레임워크, 컴포넌트, 워크플로우 가변성, .NET 프레임워크
Enterprise Framework, Component, Workflow Variability, .NET Framework

* 정평모비컴(주) 대표이사
** 바산네트워크(주)
***숭실대학교 대학원 컴퓨터학과 주임교수

1. 서 론

컴포넌트 기반 개발(Component Based Development, CBD)은 미리 구현된 블록단위의 컴포넌트를 사용하여 소프트웨어 개발 비용 및 시간을 단축할 수 있다[1][2]. 또한 컴포넌트는 내부의 상세 부분을 숨기고 인터페이스를 제공하여 대형 어플리케이션을 개발하는데 복잡성을 감소시킬 수 있다[3]. 엔터프라이즈 프레임워크 환경에서 컴포넌트 재사용과 컴포넌트 가변성을 이용하면 엔터프라이즈 어플리케이션 개발의 복잡성을 감소시키고 개발 생산성을 높일 수 있다.

현재 컴포넌트의 재사용성 및 가변성 적용을 위한 방법으로 많이 제시되고 있는 기법은 프로덕트 라인(Product Line)에서 사용하는 공통성(Commonality) 및 가변성(Variability) 분석 기법으로, 엔터프라이즈 어플리케이션 개발의 컴포넌트 구성 시 다양한 패밀리 멤버(Family Member)의 요구사항을 분석하여 반영하는 것이다[4]. 이러한 패밀리 멤버의 요구사항은 컴포넌트 내에서 업무 워크플로우 형태로 공통성 및 가변성을 포함하는 형태로 표현될 수 있으나, 다양한 패밀리 멤버(Family Member)의 요구사항에 특화된 워크플로우를 하나의 컴포넌트에 포함하거나 이를 관리하는 것은 매우 어렵다. 따라서 패밀리 멤버(Family Member)를 위한 컴포넌트간 워크플로우 가변성 관리 및 구현 방법에 대한 필요성은 끊임없이 요구되어 왔다.

본 논문에서는 .NET 기반 하에 엔터프라이즈 프레임워크에서 컴포넌트 워크플로우 가변성을 위한 구현 기법을 제시한다. 제시된 컴포넌트 워크플로우 가변성 구현 기법은 엔터프라이즈 프레임워크에서 패밀리가 사용할 수 있는 공통 업무 워크플로우가 컴포넌트 내에 이미 생성되었음을 가정하며, 다양한 패밀리 멤버에 특화된 컴포넌트간의 워크플로우 구성 및 실행을 위한 기법으로 컴포넌트간 워크플로우의 확장성을 높여주게 된다.

본 논문의 구성은 2장에서 컴포넌트 가변성 및 워크플로우에 대한 관련 연구를 제시한다. 3장에서는 .NET 기반의 컴포넌트 워크플로우 관리 시스템 아키텍처를 설계하고, 4장에서는 .NET 기반의 컴포넌트 워크플로우 관리 시스템 아키텍처를 구현한다. 6장에서는 논문에서 제시한 컴포넌트 가변선 관리를 위한 .NET 기반의 워크플로우 관리 시스템을 평가하며, 마지막으로 7장에서는 결론을 제시한다.

2. 관련 연구

2.1. 컴포넌트 가변성

컴포넌트 가변성 유형 및 범위에 대한 정형적 모델 연구에서는 컴포넌트 내부를 변경하지 않고 인터페이스 수준에서 컴포넌트를 변경하여 기능을 확장할 수 있는 컴포넌트 가변성 유형을 제시한다[5]. 이 연구에서 제시된 컴포넌트 가변성은 논리 가변성(Logic Variability), 속성 가변성(Attribute Variability), 데이터 영구 가변성(Data Persistent Variability) 그리고 워크플로우 가변성(Workflow Variability)의 네 가지 유형으로

분류된다.

먼저 로직 가변성은 여러 패밀리 멤버에서 동일한 기능을 수행하지만 기능 수행을 위한 알고리즘 또는 로직이 각 멤버에 맞게 유동적인 가변성을 의미한다. 만일, 여러 패밀리 멤버에서 동일한 기능을 수행하지만 기능 수행에 요구되는 요소의 종류, 타입 또는 개수가 가변적일 경우 이를 속성 가변성이라 한다. 데이터 영구 가변성은 컴포넌트의 데이터 영구 스키마가 어플리케이션의 데이터베이스 스키마와 물리적으로 동일하지 않을 경우를 말한다. 워크플로우 가변성은 여러 어플리케이션에서 동일한 목적을 수행하지만 목적 수행을 위한 컴포넌트 내부 또는 외부의 메시지 흐름이 각 패밀리 멤버에 따라 특화되어 흐름의 순서가 변경되는 가변성을 지칭한다.

본 논문에서는 컴포넌트 가변성 유형 및 범위에 대한 정형적 모델 연구의 워크플로우 가변성의 정의에 기반하여 패밀리에게 공통적이며 각 패밀리 멤버에 따라 가변적인 워크플로우를 관리하기 위한 컴포넌트간 워크플로우 가변성 구현 기법을 제시하게 된다.

2.2 CBD 방법론

Catalysis 방법론은 요구사항 분석부터 설계 및 구현까지 컴포넌트의 재사용 및 확장성을 위한 컴포넌트 개발 프로세스와 산출물을 제시한다[6]. 이는 설계부터 구현까지의 추적성이 보장되며, 모든 산출물이 재사용 가능하다. Catalysis 방법론에서 컴포넌트의 공통적인 기능을 제공하는 상위 인터페이스와 특화를 기능을 관리하기 위한 하위 인터페이스가 제시되며, 컴포넌트 가변성을 설정을 위한 방법은 패밀리 멤버에 맞는 특정 기능을 상위 기본 클래스를 상속받아 확장하거나 하위 인터페이스를 통해 플러그인 하는 메커니즘을 적용하게 된다. 제시된 방법론은 플러그인 기법을 통해 특화 컴포넌트를 패밀리 멤버에 맞게 설정할 수 있지만, 컴포넌트간의 워크플로우 가변성에 대한 구체적인 지침 및 구현 방법이 요구된다.

Katharine 기법은 재사용성을 위한 컴포넌트 개발 원리와 방법을 제시한다[7]. 이는 컴포넌트와 인터페이스 유형을 아키텍처 계층별로 정의하였으며, 인터페이스에 대한 정의 및 특화 인터페이스를 통한 가변성 설정 지침을 제공한다. 이때, 블랙박스 컴포넌트의 customization 인터페이스는 가변성을 설정하는 인터페이스로, 가변성이 설정되는 가변점에 따라 위임, 매개변수화, 상속 등의 특화 프로세스를 제시한다. Katharine에서 제시된 기법은 컴포넌트 가변성 설정 방법에 대한 분류가 세부적이다. 그러나 컴포넌트 가변성을 관리하고 구현하는 세부적인 절차 및 방법이 필요하다.

Componentware 프로세스는 분석에서 비즈니스 설계, 기술 설계, 스펙 및 구현단계까지의 단계를 패턴에 따라 여러 형태로 조합하여 다양한 프로세스로 구성할 수 있다[8]. 이 프로세스에서 제시하는 컴포넌트 가변성 설정을 위한 방법은 Catalysis나 Katharine에서 제시된 가변성 설정 인터페이스와 유사한 Import 인터페이스를 정의하여 요구되는 기능을 컴포넌트 외부에서 재공받을 수 있다. 그러나 Componentware 프로세스 역시 컴포

넌트 가변성 설계나 구현에 대한 구체적인 기법이나 절차가 요구된다.

앞서 제시한 Catalysis나 Katharine, Componentware와 같은 CBD방법론은 컴포넌트 워크플로우 가변성을 설정하기 위한 지침이나 구현기법이 보다는 개념적인 적용방법만을 소개하고 있다. 따라서 본 연구에서는 컴포넌트 워크플로우 가변성을 설정하기 위한 구체적인 절차와 구현기법에 대해 고려해 본다.

3. 워크플로우 가변성을 위한 아키텍처 설계

본 장에서는 워크플로우 관리를 위한 시스템 아키텍처 및 워크플로우 가변성을 위한 .NET 프레임워크의 아키텍처를 설계한다. 워크플로우 관리를 위한 시스템 아키텍처는 기존의 접근 행위자에 따라 워크플로우를 관리하였던 워크플로우 관리 시스템 연구를 바탕으로 한다[9]. 단, 기존 연구에서 접근하는 행위자에 따른 워크플로우 구성이 아닌 패밀리 멤버의 워크플로우 요구사항을 정의하여 특화된 워크플로우를 구성할 수 있으며, 또한 해당 워크플로우 단계의 실행 단위인 컴포넌트 또는 어플리케이션 정보를 검색하여 워크플로우 실행 단계에 적절히 조합될 수 있도록 발전시켰다. .NET 프레임워크 아키텍처는 .NET 프레임워크에서 지원되는 메커니즘을 기반으로 워크플로우 관리를 위한 시스템 아키텍처의 워크플로우 가변성을 처리하는 핵심 부분을 설계하였다.

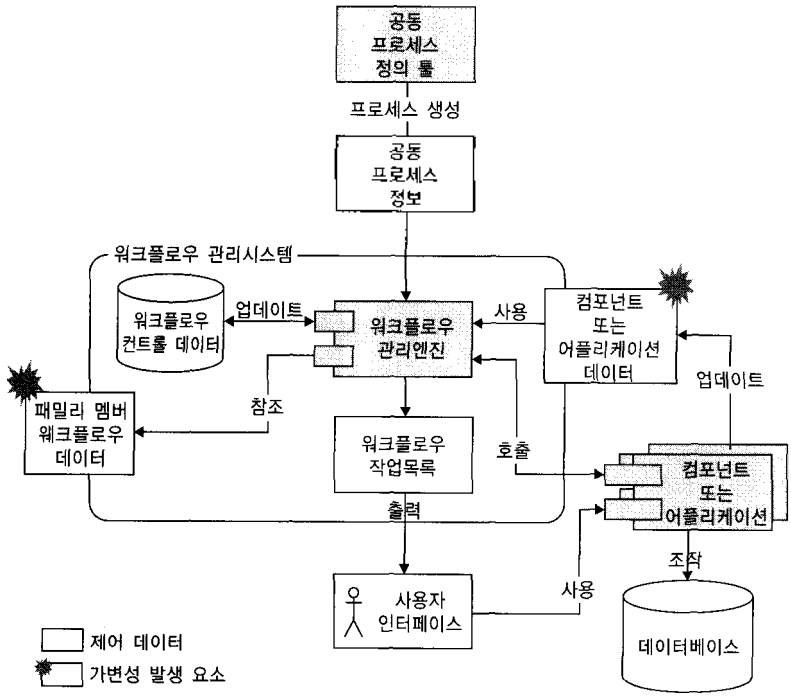
3.1. 워크플로우 관리를 위한 시스템 아키텍처

〈그림 1〉은 컴포넌트 워크플로우 가변성을 위한 외부시스템간의 관계 및 내부 시스템의 컴포넌트간 워크플로우를 나타내는 아키텍처이다. 시스템 아키텍처의 중심은 공통 프로세스를 산출물로 입력 받아 가변적 요소를 반영하여 가변 프로세스를 생성하는 워크플로우 관리시스템 내의 워크플로우 관리엔진에 있다.

〈그림 1〉에 대한 설명은 다음과 같다. 프로세스정의 툴은 패밀리의 공통적인 프로세스를 생성하는 툴이며, 이로 생성된 공통 프로세스의 산출물은 프로세스 정보로 표현되어 워크플로우 관리시스템 범위에 존재하는 워크플로우 관리엔진으로 입력된다.

워크플로우 관리엔진은 입력되는 패밀리의 공통 프로세스에 각 패밀리 멤버에 따라 특화될 수 있는 워크플로우를 가변적으로 생성하고 이를 관리, 실행할 수 있게 된다. 즉, 워크플로우 엔진은 패밀리 멤버가 요구하는 워크플로우의 정보를 나타내는 패밀리 멤버 워크플로우 데이터에 따라 워크플로우를 구성하기 위한 이미 만들어진 컴포넌트나 외부 어플리케이션의 작업의 정보를 찾아서 모아주는 컴포넌트 또는 어플리케이션 관련 데이터를 사용하여 워크플로우를 다양하게 구성할 수 있게 된다.

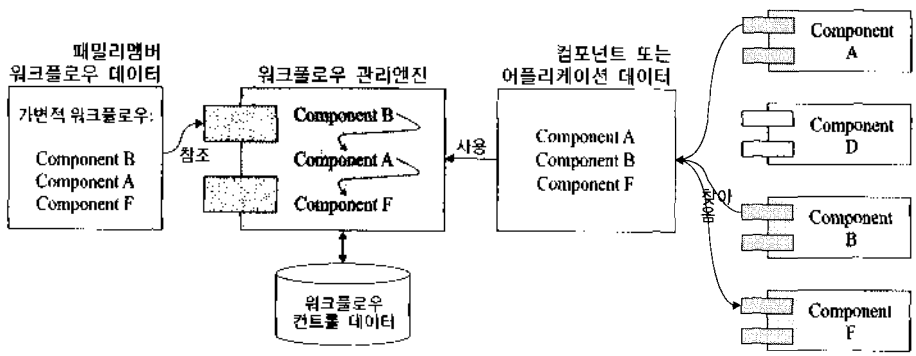
워크플로우 관리엔진에서 워크플로우를 구성하는 컴포넌트 또는 어플리케이션간의 워크플로우를 실행할 때마다 발생되는 데이터는 워크플로우 컨트롤 데이터에 임시적으로 저장되게 된다. 또한 워크플로우 관리엔진은



〈그림 1〉 워크플로우 관리를 위한 시스템 아키텍처

워크플로우를 실행하기 위해 이미 생성된 컴포넌트 또는 어플리케이션을 호출하게 되며, 반대로 어플리케이션이 실행 시 워크플로우 흐름을 변경하기 원할 경우에는 워크플로우 관리엔진을 호출할 수 있다. 워크플로우의 실행 목록은 워크플로우 작업목록에 기록되어 패밀리 멤버가 이를 확인할 수 있다.

〈그림 2〉에 의하면 각 패밀리 멤버에 따라 가변적으로 요구되는 패밀리 멤버 워크플로우 데이터를 참조하여, 워크플로우를 구성하



〈그림 2〉 워크플로우 관리엔진 메시지 흐름

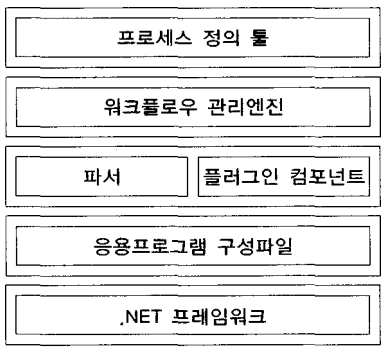
게 되는 이미 구현된 컴포넌트 또는 어플리케이션의 데이터를 찾아서 집합시킨 컴포넌트 또는 어플리케이션 데이터를 사용하여 워크플로우 관리엔진에서 가변적인 워크플로우를 구성하게 된다. 워크플로우를 수행 시 생성되는 데이터는 워크플로우 컨트롤 데이터에 임시 저장된다. 본 논문에서는 각 패밀리 멤버에 따라 이미 생성된 컴포넌트 또는 외부 어플리케이션을 조합하여 가변적인 워크플로우를 생성하는 워크플로우 관리엔진에 대한 설계 및 구현 기법을 제시하게 된다.

3.2 가변성 메커니즘의 .NET 프레임워크

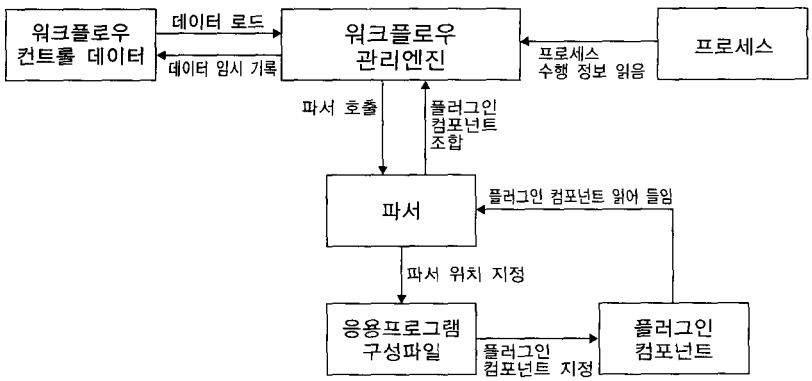
.NET 프레임워크는 〈그림 3〉과 같이 응용프로그램 구성파일(Application Configuration File)을 제공하여 동적으로 도메인에 맞는 컴포넌트를 찾아 플러그인 시키는 기반을 제공한다[10]. 이때, 워크플로우 관리엔진은 응용프로그램 구성파일에서 정의된 프로세스를 이용하고 제어하는 역할을 한다.

〈그림 2〉에 대한 설명은 다음과 같다. 프로세스 정의 툴은 〈그림 1〉에서와 같이 패밀리의 공통적인 기능을 위한 프로세스를 정의하였으며, 워크플로우 관리엔진은 각 패밀리 멤버에 따라 가변적인 프로세스를 생성하는 역할을 한다. 이때, 프로세스 정의 툴은 공통적인 프로세스를 패밀리가 정의한 XML 등의 자유로운 형태로 자동 생성하여 워크플로우 관리엔진에서 관리할 수 있으며, 가변적인 프로세스 또한 각 패밀리 멤버가 자유로운 형태로 직접적으로 정의하여 워크플로우 관리엔진에서 참조할 수 있게 할 수 있다. 이러한 워크플로우의 정보는 응용프로그램 구성파일로 전달하여 워크플로우를 구성하는 컴포넌트들을 조합하게 한다.

〈그림 4〉와 같이, 워크플로우 관리엔진은 가변적인 워크플로우를 구성하는 컴포넌트를 플러그인하기 위해 현재 위치에서 이미 만들어진 특정 서비스를 제공하는 플러그인 컴포넌트를 파서를 통해 찾게 된다. 이때, 응용프로그램 구성파일은 프로세스 단계에 해당하는 컴포넌트 정보를 찾는 파서와 도메인에서 요구하는 클래스를 찾아 플러그인 시켜주는 역할로 파서 및 플러그인 컴포넌트 정보를 가지고 있으며, XML 형태로 되어 있다.



〈그림 3〉 가변성 메커니즘의 .NET 프레임워크



〈그림 4〉 가변성 메커니즘의 흐름 아키텍처

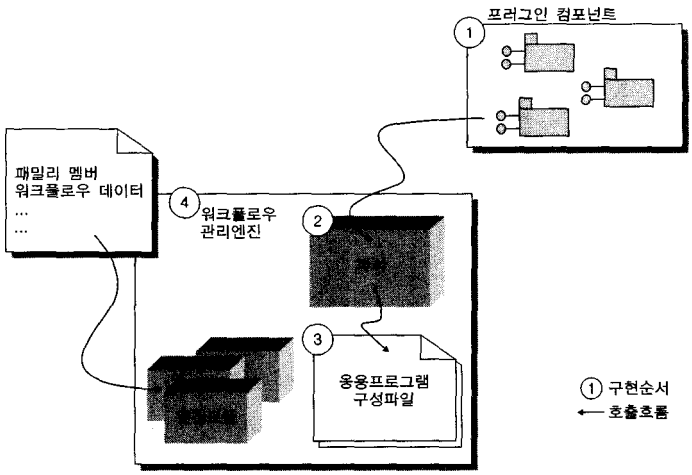
지금까지 제시된 워크플로우 가변성을 위한 아키텍처에서 제시된 것처럼, 본 논문에서는 기존의 도메인의 업무 특성에 맞게 컴포넌트 내의 워크플로우를 변경하던 기법에서 컴포넌트 간의 워크플로우를 변경하는 아키텍처를 제시하였다. 따라서, 도메인의 패밀리 멤버에 따라 특화된 워크플로우를 구성하기 위해 프로세스 단계마다 요구되는 이미 생성된 컴포넌트 또는 어플리케이션을 재사용하여 워크플로우를 구성하게 된다. 또한 패밀리의 공통적인 워크플로우가 이미 정의되었으며, 패밀리 멤버에 맞는 워크플로우를 일부 생성하거나 제외시킬 수 있으므로 엔터프라이즈 프레임워크의 어플리케이션 개발 생산성 효율을 극대화 시킬 수 있다.

4. .NET에서의 워크플로우 가변성 구현

본 논문에서 제안하는 워크플로우 가변성은 워크플로우 관리엔진을 위한 .NET 프레임워크의 플러그인 메커니즘을 적용한다. 플러그인 기법을 위해서는 .NET 기반의 응용프로그램 구성파일(application configuration file)을 이용하면 컴포넌트간 워크플로우를 동적으로 제어할 수 있다. 〈그림 6〉은 워크플로우 관리엔진 구현을 위한 흐름도를 나타낸다.

〈그림 6〉에서와 같이, 워크플로우 관리엔진은 패밀리 멤버에 맞는 워크플로우 흐름이 작성된 외부 파일인 패밀리 멤버 워크플로우 데이터를 읽어 들여 프로세스를 구성하는 컴포넌트들의 플러그인 순서를 정의하게 된다.

워크플로우 관리엔진이 컴포넌트를 플러그인 하여 패밀리 멤버에 특화된 워크플로우를 구성하기 위해서는 먼저 .NET 프레임워크에서 제공하는 제안 기준에 따라 플러그인 컴포넌트(①)를 구현하게 된다. 또한 워크플로우 관리엔진 내에서는 플러그인 준비가 된 컴포넌트를 읽어 들이기 위한 파서(②)가 구현되어야 한다. 플러그인 컴포넌트와 파서가 구현이 되었다면, 파서의 위치를 지정하고 또한 그에 따른 플러그인 컴포넌트를 선정하는 응용프로그램 구성파일(③)을 구현한다. 마지



〈그림 5〉 워크플로우 관리엔진 구현 흐름도

막으로 패밀리 멤버 워크플로우 데이터에 따라 워크플로우의 흐름을 구성하고, 가변적인 워크플로우를 실행하기 위해 필요한 환경을 지원하는 워크플로우 관리엔진(④)을 구현하게 된다.

4.1 단계 1: 플러그인 될 컴포넌트 생성

워크플로우 가변성을 위해 플러그인 되는 컴포넌트는 〈그림 6〉과 같이 .NET 프레임워크에서 지원해주는 IPlugin라는 공통 인터페이스를 상속받게 된다. IPlugin 인터페이스를 상속받는 컴포넌트는 워크플로우를 구성하기 위해 플러그인 될 수 있는 플러그인 컴포넌트의 속성을 가지게 된다.

4.2 단계 2: 파서 생성

워크플로우를 구성하는 컴포넌트를 읽어오기 위한 파서는 .NET 프레임워크에서 제공하는 System.Configuration.IConfigurationSectionHandler 인터페이스를 상속받는다. 이는 플러그인 될 컴포넌트 정보가 기록된 파일을 읽어 들여서 해당 객체를 반환하게 된다[10]. 〈그림 7〉은 파서를 위해 .NET 프레임워크에서 사용할 수 있는 인터페이스의 멤버를 나타내고 있다. 이때, 이 인터페이스 멤버는 Create 함수를 가지고 있는데, 이를 오버라이드 하여 구현하게 되면 파서의 속성이 생기게 된다.

파서 인터페이스를 상속받아 생성된 파서

```
public interface IPlugin
{
    string Name{get;}
    bool Exceute(string info);
}
```

〈그림 6〉 플러그인 공통 인터페이스

```
public interface IConfigurationSectionHandler
{
    public object Create(object parent,
                         object configContext, System.Xml.XmlNode section);
}
```

〈그림 7〉 파서 인터페이스

는 플러그인 객체를 읽어 들인 후, 워크플로우 관리엔진에 필요한 객체들을 모으는 임의의 클래스를 만들어 플러그인 컴포넌트를 쉽게 관리하게 한다.

4.3 단계 3: 응용프로그램 구성파일 생성

.NET 프레임워크에서 플러그인 기법을 쉽게 구현하기 위해서는 응용프로그램 구성파일 장치를 사용한다. 응용프로그램 구성파일은 플러그인 정보를 설정하는 파일로, XML 형식의 파일을 통해서 파서의 위치와 플러그인 될 컴포넌트를 지정하게 된다.

〈그림 8〉은 응용프로그램 구성파일의 형식을 보여준다. 그림 8의 (1)은 플러그인 할 클래스를 읽어 들이는 파서 클래스의 위치를 지정하는 것이며, (2)는 플러그인 되는 컴포넌트를 지정할 수 있게 된다. 즉, domain.domainTask에 위치하는 파서가 ComponentA와 ComponentC 등의 플러그인 컴포넌트를 읽어 들일 수 있도록 설정하게 된다.

4.4 단계 4: 워크플로우 관리엔진 생성

워크플로우 관리엔진은 워크플로우 실행 순서가 기록된 파일을 읽어 도메인와 패밀리 멤버에 맞는 가변적인 워크플로우를 실행하게 된다. 즉, 〈그림 5〉의 패밀리 멤버 워크플로우 데이터의 정보만 변경하면 동적으로 컴포넌트간 워크플로우를 변경할 수 있는 것이다. 또한 워크플로우 관리엔진은 워크플로우를 실행하는 것 외에 실행시의 작업 목록을 생성하여 사용자에게 출력을 해주거나, 실행 시간을 체크하는 등의 워크플로우 실행 시 요구되는 환경을 제공하게 된다.

워크플로우 관리엔진을 구현하기 위해서는 두 가지 방법을 적용할 수 있다. 첫 번째 방법은 워크플로우 관리엔진 자체 내에 워크플로

```
<configuration>
<configSections>
      <section name="plugins"
            type="domain.domainTask, PluginParser" /> ... (1)
</configSections>
   <plugins>
      <plugin type="domain.PlugIn.ComponentA, LoanPlugin" />
      <plugin type="domain.PlugIn.ComponentC, LoanPlugin" />
      . . .
   </plugins> ... (2)
</configuration>
```

〈그림 8〉 응용프로그램 구성파일

```
public class WorkflowEngine
{
   Private PluginCollectoin plugins = null;
   . . .
   public void GetPluginComponent() ... (1)
   {
      domainTask task = new domainTask();
      plugins         = task.Create();
   }
   public bool executeWorkflow() ... (2)
   {
      using (StreamReader sr = new StreamReader("WorkflowFile.txt")) ... (3)
      {
          String line;
          while ((line = sr.ReadLine()) != null)
          {
    . . .
          }
   . . .
}
```

〈그림 9〉 워크플로우 관리엔진 일부 코드

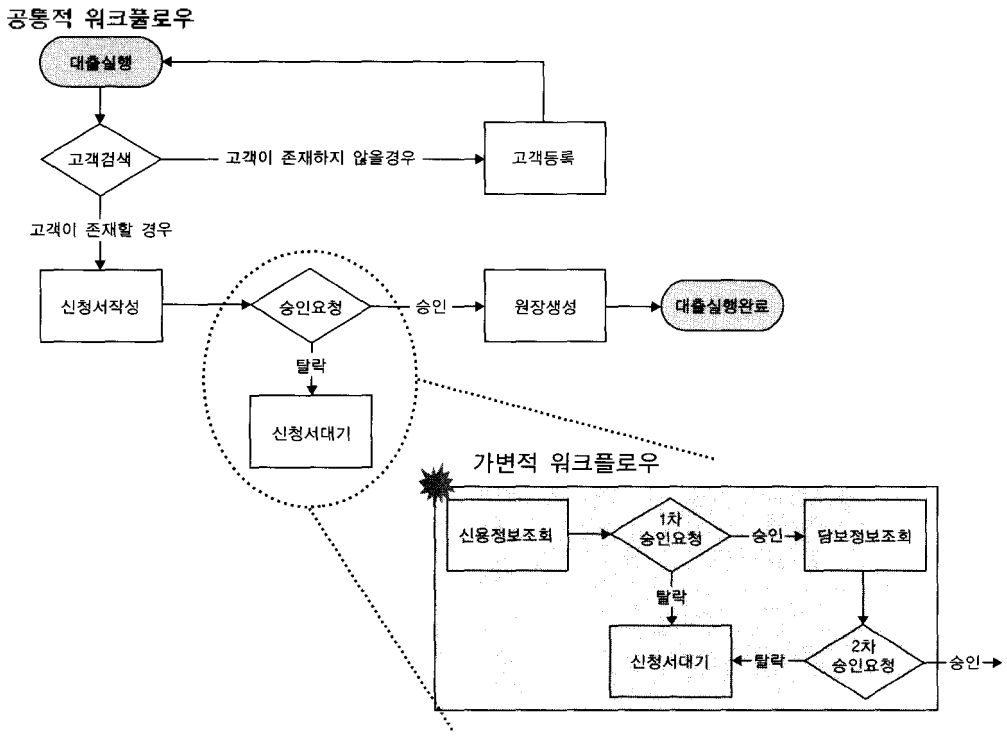
우의 흐름이 적힌 파일을 직접 읽어 들어 흐름을 직접 실행하는 방법이 있고, 두 번째 방법은 리플렉션(Reflection) 기능을 이용하여 워크플로우를 제어하는 클래스를 동적으로 생성하는 방법이 있다. 본 논문에서는 첫 번째 방법인 워크플로우 관리엔진에 직접 구현하는 방법을 보여주고자 한다. 이는 첫 번째 방법이 두 번째 방법의 기초가 되기 때문이다. 〈그림 9〉는 워크플로우 관리엔진의 일부 코드이다.

〈그림 9〉의 (1)은 워크플로우를 구성하기 위해 플러그인 될 클래스를 생성하는 부분이다. (2)는 (3)에서와 같이 워크플로우 순서가 기록된 workflowFile.txt 형식의 파일을 읽어 들여 도메인에 맞는 가변적 워크플로우를 실행하게 된다. 이때, 순서가 기록된 파일은 패밀리 멤버의 특색에 맞게 파일의 형식을 지정할 수 있다. 워크플로우 관리엔진의 보여지지 않은 나머지 코드는 실행된 플로우의 목록을 생성하며 사용자에게 출력하는 등의 워크플로우 가변성을 지원하는 환경을 구현하는 부분이 된다.

5. 사례 연구

본 논문에서는 은행대출업무 도메인에 대한 간단한 워크플로우 예시와 함께 .NET 기반의 워크플로우 가변성 구현 기법을 제시한다. 은행대출업무의 요구되는 워크플로우는 다음과 같다.

〈그림 10〉에서 정의된 은행대출업무 공통 프로세스 중 대출을 승인하는 업무 흐름에서 은행에 따라 가변적인 요소가 발생하게 되었다. 즉, 한번의 대출 승인요청만을 처리하는 공통 프로세스를 SS은행에서는 신용정보조회 처리 후 1차 승인요청이 이루어지며, 승인이 된 정보에 대한 담보정보조회가 이루어진 후 2차 승인요청이 이루어지도록 워크플로우가 확장된 것이다. 이러한 가변적인 워크플로



〈그림 10〉 대출업무 워크플로우 공통성 및 가변성

우를 관리 및 생성하는 단계는 다음과 같이 진행된다.

5.1 단계 1: 플러그인 될 컴포넌트 생성

먼저 SS은행에서 요구되는 기능들을 구현한 플러그인 컴포넌트 생성한다. 〈그림 11〉은 각 기능을 구현한 컴포넌트는 중 〈그림 6〉의 IPlugin 공통 인터페이스를 상속받아 구현한 신용정보조회 컴포넌트의 일부이다.

이외에도 SS은행에서 은행대출업무에서 워크플로우 진행에 필요하여 플러그인 되는 담보정보조회나 신청서대기 등의 컴포넌트 역시 IPlugin을 상속받아 플러그인 컴포넌트의 속성을 지니게 된다.

5.2 단계 2: 파서 생성

다음으로, 구현이 완료된 SS은행의 은행대출업무 워크플로우를 구성하는 플러그인 컴포넌트를 읽는 파서를 구현한다.

〈그림 12〉는 〈그림 7〉의 ICongiguration SectionHandler 인터페이스를 상속받아 구현한 파서의 일부이다. 〈그림 12〉의 PluginCollection은 파서를 통해 읽은 플러그인 컴포넌트들을 가지는 클래스로, 플러그인 컴포넌트를 쉽게 관리하기 위해 임의로 만든 클래스이다. 즉, SS은행에서 워크플로우 실행

시 필요한 플러그인 컴포넌트를 모아서 관리하게 된다. 단, PluginCollection 클래스 대신 .NET에서 제공하는 Collection 형으로 대체할 수 있다. 예를 들면 Arraylist 같은 형이다.

5.3 단계 3: 응용프로그램 구성파일 생성

전 단계에서 플러그인 컴포넌트와 파서가 구현되었다면 이를 연결해 주는 응용프로그램 구성파일을 작성한다. <그림 13>은 <그림 12>의 SS은행에서 사용되는 LoanTask 파서와 <그림 11>의 신용정보조회(SearchCreditInfo) 컴포넌트를 연결해 주는 응용프로그램 구성 파일 예이다.

```
namespace ezLoan.PlugIn
{

      [
       JustInTimeActivation(true),
       ComponentAccessControl,
       SecurityRole("관리자"),
       SecurityRole("일반사용자",
       SetEveryoneAccess = true),
       Transaction(TransactionOption.Supported)
]

     public class SearchCreditInfo : IPlugin, ServicedComponent
     {
         public ValidateCustomerInfo() { }

         [ AutoComplete ]
         public bool Exceute(string info)
         {
          // 로직구현
         }
     }
}
```

<그림 11> 플러그인 컴포넌트 예제

```
public class LoanTask : IConfigurationSectionHandler
{

   public object Create(object parent, object configContext,
                                          System.Xml.XmlNode section)
   {
      PluginCollection plugins = new PluginCollection ();

      foreach(XmlNode node in section.ChildNodes)
      {
          object plugObject = Activator.CreateInstance
                              (Type.GetType(node.Attributes["type"].value));
          IPlugin plugin = (IPlugin)plugObject;
          plugins.Add(plugin);
      }
      return plugins;
   }
}
```

<그림 12> 파서 구현 코드 예제

```
<configuration>
<configSections>
      <section name="plugins"
            type="ezLoan.LoanTask, PluginParser" />
</configSections>
   <plugins>
      <plugin type="ezLoan.PlugIn.SearchCreditInfo, LoanPlugin" />
      . . .
   </plugins>
</configuration>
```

〈그림 13〉 응용프로그램 구성파일 예제

5.4 단계 4: 워크플로우 관리엔진 생성

마지막으로 은행마다 특화된 워크플로우를 실행시키기 위해 워크플로우 관리엔진을 생성한다. 〈그림 14〉는 은행대출업무 워크플로우를 실행시키기 위한 워크플로우 엔진의 일부 코드이다. executeWorkflow()함수에서 나타난 것처럼 SS은행의 요구하는 워크플로우의 순서는 자유로운 형식으로 WorkflowFile에 기록되며, 워크플로우 관리엔진은 이를 읽어 들여 워크플로우 각 단계마다 해당 플러그인 컴포넌트를 실행하게 된다.

〈그림 14〉 워크플로우 관리엔진 일부 코드

또한 〈그림 14〉의 워크플로우 관리엔진은 워크플로우를 실행시키는 것 외에 은행의 요구사항에 따라 각 단계를 실행시킬 때마다 실행되는 컴포넌트의 이름이 출력되게 되어 있다.

지금까지 은행대출업무를 각 은행에 따라 컴포넌트를 새로 구현하는 것이 아니라 기존의 만들어진 컴포넌트간 작업 순서를 .NET 기반의 워크플로우 관리엔진을 통해 쉽게 변경할 수 있는 구현기법을 살펴보았다.

6. 평 가

본 논문에서 제시한 컴포넌트 워크플로우 가변성 구현 기법을 기존 Catalysis 방법론, Katharine 기법과 Componentware 프로세스와 비교 평가하면 〈표 1〉과 같다.

〈표 1〉에서 본 논문의 기법과 비교 평가되는 기존의 연구방법은 컴포넌트 가변성을 설정하기 위해 가변성 인터페이스를 사용하여 플러그인 메커니즘을 제공하거나 기본 클래스를 상속하여 확장하는 개념을 도입하였다. 따라서 컴포넌트 내부에 설정된 속성을 변경하는 속성 가변성은 지원이 되지 않으며, 플러그인 메커니즘 등을 통하여 로직 가변성은 지원이 가능하다. 또한 기존의 연구에서는 컴포넌트 실행 시 필요한 다른 컴포넌트 또는 클래스를 플러그인 할 수는 있으나 플러그인 되는 컴포넌트의 흐름을 제어할 수 없었다. 그러나 본 연구에서는 컴포넌트 간의 흐름을 제어하는 기법을 제시하여 각 패밀리 멤버에 따라 특화되는 워크플로우 가변성을 제공할 수 있다.

블랙박스 설계는 가변성 설정 시 컴포넌트

```
public class LoanScheduler
{
   Private PluginCollectoin plugins = null;
   .
   .
   .
   public void GetTask()
   {
      LoanTask task = new LoanTask();
      plugins       = task.Create();
   }
   public bool executeWorkflow()
   {
       try
       {
           using (StreamReader sr = new StreamReader("WorkflowFile.txt"))
           {
               String line;
               while ((line = sr.ReadLine()) != null)
               {
        foreach(IPulgIn plugin in plugins)
        {
  if(line.Equals(plugin.name)
  {
                  plugin.Execute();
     console.WriteLine(¡'execute ¡' + line);
     break;
  }
                    }
                }
  return true;
            }
            return false;
        }
        catch (Exception e)
        {
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
            return false;

        }
   }
   .
   .
}
```

〈그림 14〉 워크플로우 관리엔진 일부코드

내부의 정보가 외부에 누출되지 않는 것을 말하므로, 본 논문의 기법뿐만이 아니라 모든 기존 연구에서 지원되는 부분이다.

〈표 1〉의 평가 요소 중 확장성이란 컴포넌트 또는 어플리케이션의 개발이 완료된 후에 추가적으로 가변적인 요소를 설정 또는 변경할 수 있는가에 대한 평가요소로, 기존 연구에서는 정적 모델을 설계하는 컴파일 타임에 가변성을 설정하게 되므로 개발 완료 이후 가변적 요소를 설정하기가 어렵다. 그러나 본 논문에서 제시하는 기법은 컴파일 타임뿐만이 아니라 런 타임에서도 워크플로우가 기록된 외부 파일을 변경하여 다양한 워크플로우를 실행시킬 수 있게 된다.

〈표 1〉 컴포넌트 워크플로우 가변성 구현 기법 비교

○: 가능, △: 보통, ×: 불가능

평가요소 \ 연구		본 논문의 구현기법	Catalysi	Katharin	Componentwar
속성 가변성		×	×	×	×
로직 가변성		○	○	○	○
워크플로우 가변성	컴포넌트 내	×	×	×	×
	컴포넌트 간	○	×	×	×
블랙박스 설계		○	○	○	○
확장성		○	×	×	×
동적 바인딩		○	△	△	△
재사용성		○	△	△	△
습득 용의성		△	△	△	△
상세구현지침제공		○	△	×	△

평가 요소의 재사용성이란, 제시된 연구방법을 사용하여 엔터프라이즈 프레임워크 환경에서 어플리케이션을 개발하였을 경우에 시스템에 대한 재사용성을 말한다. 기존 연구에서는 기능을 지원하는 컴포넌트만을 재사용할 수 있으나, 본 연구에서 제시된 기법은 플러그인 컴포넌트뿐만 아니라 워크플로우 관리엔진을 통하여 생성된 프로세스까지도 다시 사용할 수 있게 된다.

단, 기존 연구뿐만 아니라 본 논문의 기법에서도 가변성을 설정하기 위한 플러그인 컴포넌트 설계 또는 구현 기법이나 외부 파일의 형식에 대한 사전 지식이 필요할 수 있다. 그러나 본 연구에서는 기존 연구보다 워크플로우 가변성을 설정하기 위한 상세한 구현 지침을 제공하여 .NET 기반의 워크플로우 가변성 구현 기법 도입이 용이할 것이다.

7. 결 론

본 논문에서는 기존의 도메인의 업무 특성에 맞게 컴포넌트 내의 워크플로우를 변경하던 기법에서 컴포넌트 간의 워크플로우를 변경하는 아키텍처를 제시하였으며, .NET 프레임워크를 기반으로 워크플로우 가변성을 구현하는 기법을 제안하였다. 제시된 기법은 패밀리 멤버가 요구하는 프로세스에 따라 워크플로우 가변성을 지원하는 워크플로우 관리엔진을 통하여 해당하는 프로세스 단계의 위치에서 필요한 플러그인 컴포넌트들을 찾아 이들을 조합한다. 따라서 패밀리 멤버에 특화된 워크플로우를 생성할 수 있게 된다.

또한 제시된 워크플로우 가변성 구현 기법은 워크플로우 가변성을 생성할 때, 컴포넌트를 워크플로우에 맞게 플러그인 하거나 또는

워크플로우 관리엔진을 통하여 이미 생성된 프로세스를 다시 사용하는 등의 높은 재사용성을 지원하게 된다. 또한 어플리케이션의 설계 단계뿐만 아니라 개발 완료 이후에도 워크플로우가 기록된 외부 파일을 변경하여 다양한 워크플로우를 실행시킬 수 있는 등의 높은 확장성을 기대할 수 있다.

따라서 본 논문을 통해 패밀리의 공통 요구사항을 반영하고 패밀리 멤버에 맞는 워크플로우를 생산하는 엔터프라이즈 프레임워크의 어플리케이션 개발 생산성 효율을 극대화 시킬 수 있을 것이다.

참 고 문 헌

[1] Heineman, G. T., Council, W. T., Component based Software Engineering, Addison Wesley, 2001.

[2] D'Souza D., Wills A., Object, Components, and Frameworks with UML, And Frameworks with UML, Addison Wesly, 1999.

[3] Short, K., Component Based Development and Object Modeling, Sterling Software, Technical Handbool Version 1.0, Februaray 1997.

[4] Weiss D. M., and Lai C.T.R., Software Product Line Engineering: A Family Based Software Engineering Porcess, Addison Wesley, 1999.

[5] Dong Seob So, Seok kyoo Shin and Soo Dong Kim, "A Formal Model of Component Variability Types and Scope", Journal of KISS: Software and Applications, Vol. 30, No.05, pp. 414-429, June 2003.

[6] DSouza, D. S. and Wills, A. C., Objects, Components, and Frameworks with UML: The Catalysis Approach, Addison Wesley, 1999.

[7] Whitehead, K., Component-based Development: Principles and Planning for Business Systems, Addison Wesley, 2002.

[8] Rausch A., "Software Evolution in COMPONENTWARE Using Requirements/Assurances Contracts", Proceedings of the 22th International Conference on Software Engineering, June 2000.

[9] Gimenes, M. S., Tanaka, S. and Oliverira, J., "An Object Oriented Framework for Task Scheduling", Proceedings of the TOOLS Europe, June, 2000.

[10]The MSDN Library, http://msdn.microsoft.com.

저 자 소 개

노재우 (E-mail : rjw@krjmc.com)
1984. 2 숭실대학교 전자계산학과(공학사)
1993. 8 숭실대학교 정보과학대학 정보산업학과(공학석사)
2001. 8 숭실대학교 대학원 컴퓨터학과(박사)
1984. 4 ~ 2003. 12. 정평모비컴(주) 대표이사
2004. 1 ~ 현재 정평모바컴(주) 대표컨설턴트
관심 분야 객체지향소프트웨어공학, 소프트웨어 재사용,
컴포넌트 기반 소프트웨어공학, 닷넷 시스템 개발 방법론

이승훈 (E-mail : white73@netsgo.com)
2002. 2 숭실대학교 컴퓨터학부(공학사)
2004. 3 ~ 현재 숭실대학교 대학원 컴퓨터학과 석사과정
2002. 10 ~ 현재 바산네트워크(주) 전무
관심 분야 소프트웨어 재사용, 소트프웨어 리엔지니어링,
소프트웨어 재공학/역공학, 소프트웨어 테스팅

류성열 (E-mail : syrhew@computing.soongsil.ac.kr)
1997. 2 아주대학교 컴퓨터공학부(공학박사)
1997. 3 ~ 1998. 3 George Mason University 교환교수
1998. 3 ~ 2001. 2. 숭실대학교 정보과학대학원 원장
1981. 3 ~ 현재 숭실대학교 정보과학대학 컴퓨터학부 교수
1998. 3 ~ 현재 숭실대학교 전자계산원 원장
2002. 9 ~ 현재 숭실대학교 대학원 컴퓨터학부 주임교수
관심 분야 리엔지니어링, 소프트웨어 유지보수, 소프트웨어 재사용,
소프트웨어 재공학/역공학, 소프트웨어 테스팅,
컴포넌트 기반 소프트웨어 공학