중소기업정보기술융합학회 논문지 제3권 제1호 pp. 1-6, 2013

ACME를 이용한 멀티플랫폼 지원 아키텍처 표현

박재진, 고재철, 홍장의*
충북대학교 컴퓨터과학과

Architectural Representation to Support Multi-Platform Applications Using ACME

Jae-Jin Park, Jae-Chul Ko, Jang-eui Hong*
Department of Computer Science, Chungbuk University

요 약 소프트웨어 품질이 중요시 되고 있는 최근 소프트웨어 아키텍처에 대한 연구는 활발하게 진행되고 있다. 또한 스마트 폰의 단말 플랫폼의 다양화로 인하여, 다양한 플랫폼에 탑재하는 서비스를 제공하기 위해 어플리케이션의 개발에 대한 노력이 증가되었다. 그러나 이러한 다양한 플랫폼 지원에 대한 노력을 감소시키기 위하여 다양한 솔루션들이 제시되었고, 이중의 대표적인 것이 FireMonkey 프레임워크이다. 본 연구에서는 FireMonkey 프레임워크의 아키텍처를 ACME로 표현하여, 소프트웨어 개발자가 다양한 플랫폼을 대상으로 어떻게 어플리케이션을 개발해야 하는지에 대한 가이드라인을 제공할 수 있도록 하였다. 이를 통해 소프트웨어 아키텍처가 멀티 플랫폼을 지원하기 위하여 어떠한 관점을 고려해야하는지 알 수 있게 하였다.

Abstract As the important of software quality has being emphasized, the studies on software architecture also have being performed actively. On the other hand, due to the diversification of Smartphone platforms, the effort was increased to develop the application for supporting those multiple platforms. However several solutions are suggested to reduce the effort, and a representative solution is FireMonkey framework. In this paper, we represent the FireMonkey framework using ACME which is a language to describe software architecture. Such representation can provide the guideline to develop the application for the multiple platforms. Also it supports the information of that which perspectives are critical to develop such applications.

Key Words : ACME, Multi platform, Software Architecture Perspective

1. 서론

최근 복잡하고 다양한 소프트웨어 시스템을 개발하는데 있어 높은 수준의 추상화를 나타내는 소프트웨어 아키텍처는 소프트웨어 개발에 참여하는 사람들의 의사소통과 설계 결정에 중요한 정보를 제공하고 있다[1]. 또한 소프트웨어 시스템에 요구되는 품질은 그 시스템의 소프트웨어 아키텍처에 의해 결정되기 때문에 아키텍처의 중요성은 날로 증가하고 있다[2].

오늘 날 스마트폰의 보급과 더불어 다양한 플랫폼을 대상으로 어플리케이션을 개발해야하는 경우가 많아졌다[3]. 이는 소프트웨어 개발 도구가 특정 플랫폼에 한정적이지 않은 멀티 플랫폼을 대상으로 할 수 있어야 한다는 말과 같다. 멀티 플랫폼을 지원하는 대표적인 프레임

이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업 지원을 받아 수행된 것임(2011-0010396).
접수일 : 2013년 3월 18일 수정일 : 2013년 3월 28일 게재확정일 : 2013년 4월 10일
*교신저자 : 홍장의(jehong@chungbuk.ac.kr)

워크는 FireMonkey이다[4]. FireMonkey는 Native Code를 변경 없이 그대로 사용할 수 있는 편리함이 제공하기 위해 제안된 프레임워크이다. 그림 1은 FireMonkey의 특징을 나타내고 있다. 그림 1을 보면 FireMonkey 프레임워크가 응용 개발을 C++나 델파이 언어를 그대로 사용할 수 있으며, Windows, MAC, iOS, Android 등의 모든 플랫폼을 지원할 수 있음을 보여준다.

Fig. 1. FireMonkey Framework

본 논문에서는 FireMonkey 프레임워크를 대상으로 멀티 플랫폼 아키텍처를 표현한다. 멀티 플랫폼 아키텍처를 표현하기 위하여 ADL(Architecture Description Language) 중 하나인 Acme[5]를 사용한다. Acme를 이용한 FireMonkey의 표현은 해당 프레임워크의 소프트웨어 아키텍처를 복원하는 것이며, 이러한 아키텍처의 개발을 통해 멀티 플랫폼을 지원하는 소프트웨어가 지녀야 할 속성은 무엇인가를 확인할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련된 연구들을 살펴보고 3장에서는 Acme의 구성요소를 알아본다. 4장은 Acme로 FireMonkey 아키텍처를 표현하고 5장은 멀티 플랫폼 지원 방안을 소개한다. 마지막으로 6장은 결론 및 향후 연구로써 멀티 플랫폼을 대상으로 소프트웨어를 개발할 때 고려해야할 요소들과 추후 연구할 사항에 대해 기술한다.

2. 관련연구

본 논문은 FireMonkey 아키텍처를 Acme로 표현하고자 한다. 이를 위해 소프트웨어 아키텍처를 먼저 이해하고, Acme에 대한 기존 연구를 살펴본다.

2.1 소프트웨어 아키텍처

소프트웨어 아키텍처는 소프트웨어 개발에 참여하는 사람들 간의 의사소통을 원활하게 지원하고 시스템 설계 결정에 대한 합리적인 판단을 지원하는 높은 수준의 추상화 모델이다[6]. 또한 시스템의 구성요소와 구성 요소들 사이의 연결 관계, 시스템의 설계와 진화를 통제하는 원칙과 가이드라인 등으로 구성되어 있다[7].

소프트웨어 아키텍처를 기술하는 기법은 비주얼한 모델로 나타내는 방법과 텍스트 형태의 서술방식으로 모두 표현 가능하다.

2.2 Acme

시스템의 구조를 표현하기 위해 UML과 같은 언어들을 사용할 수 있지만, 복잡하고 큰 시스템을 표현하기에는 한계가 존재한다. 아키텍처를 보다 정확하고 명확하게 설계 및 분석을 위해서는 ADL(Architecture Description Language)를 사용해야 한다. ADL 중 하나인 ACME는 다수의 ADL을 활용할 수 있도록 서로 다른 ADL로 표현된 아키텍처들을 상호변환과 공유할 수 있도록 지원하는 언어이다[8].

ACME에서는 구조, 성질, 제약사항, 타입과 스타일의 네 가지를 정의하고 있다[9]. 각각의 정의에 대해 간략히 살펴보면 다음과 같다. 구조는 컴포넌트, 커넥터, 시스템, 포트 역할, 표현, 표현 맵을 정의한다. 부가적인 정보를 표현하는 성질은 모든 디자인 요소가 가질 수 있는 정의로, 메시지 요청이나 소스코드의 위치, 상호작용 프로토콜 등을 정의할 수 있다. 또한 제약사항은 모든 디자인요소가 정의할 수 있는 부분으로 미리 정의된 predicate함수를 이용한다[10]. 마지막으로 스타일은 디자인 요소 타입의 집합으로 디자인 전문지식의 패키징, 분석 및 코드 생성 도구, 아키텍처 표준 적합성 검증 등에 이용된다[5].

3. ACME 구성 요소

ACME로 FireMonkey 아키텍처를 표현하기 위해 간단하게 ACME의 구성요소에 대해 설명한다. ACME는 아키텍처를 명세하기 위해 크게 5개의 요소로 이루어져 있고 그림 2와 같이 아키텍처를 표현할 수 있다.

컴포넌트(Component)는 시스템의 핵심적인 연산 요소와 데이터를 표현할 수 있다. 커넥터(Connector)는 컴포넌트 사이의 관계를 표현해 주고 시스템(System)은 컴포넌트와 커넥터의 환경을 표현한다. 특성(Property)은

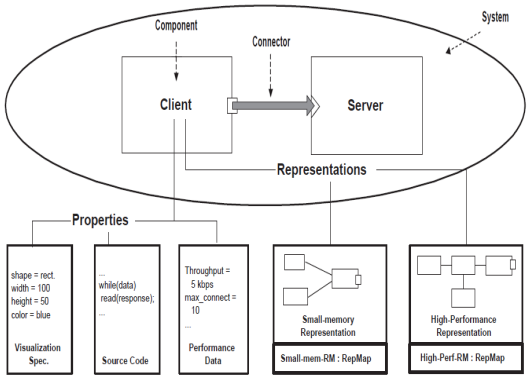


Fig. 2. Elements of an Acme Description

컴포넌트에 모두 표기할 수 없는 정보들의 표현을 지원한다. 표현 맵(RepMap)은 해당 컴포넌트가 시스템의 외부에서 어떻게 사용되는지 표현할 수 있다.

4. FireMonkey 아키텍처

기존의 FireMonkey 프레임워크의 소스 코드를 바탕으로 분석한 소프트웨어 아키텍처를 Acme로 표현해 보았다. Acme로 표현된 FireMonkey 아키텍처는 그림 3과 같다. 그림 3을 보면, 7개의 컴포넌트와 다수의 Rep-Map 및 관계가 존재하는 것을 알 수 있다. ACME를 이용하여 표현된 FireMonkey 프레임워크의 아키텍처에 대하여 구체적으로 살펴보겠다.

그림 4는 application 컴포넌트로써 어플리케이션의 시작, 종료, 그리고 진행을 관리 할 수 있다. control 컴포넌트는 그림 5와 같이 구성되며 어플리케이션 화면에 사용되는 UI 컴포넌트들을 가지고 있다. 해당 내용이 너무 방대하기 때문에 본 논문에서는 세부사항을 표현하지 않고, 추상화하여 나타내었다.

```
Component application = {
 Ports { run; terminate; }
}
```

Fig. 4. application component

```
Component control = {
  Ports { input_event_proc, draw_text, hide_app }
}
```

Fig. 5. control component

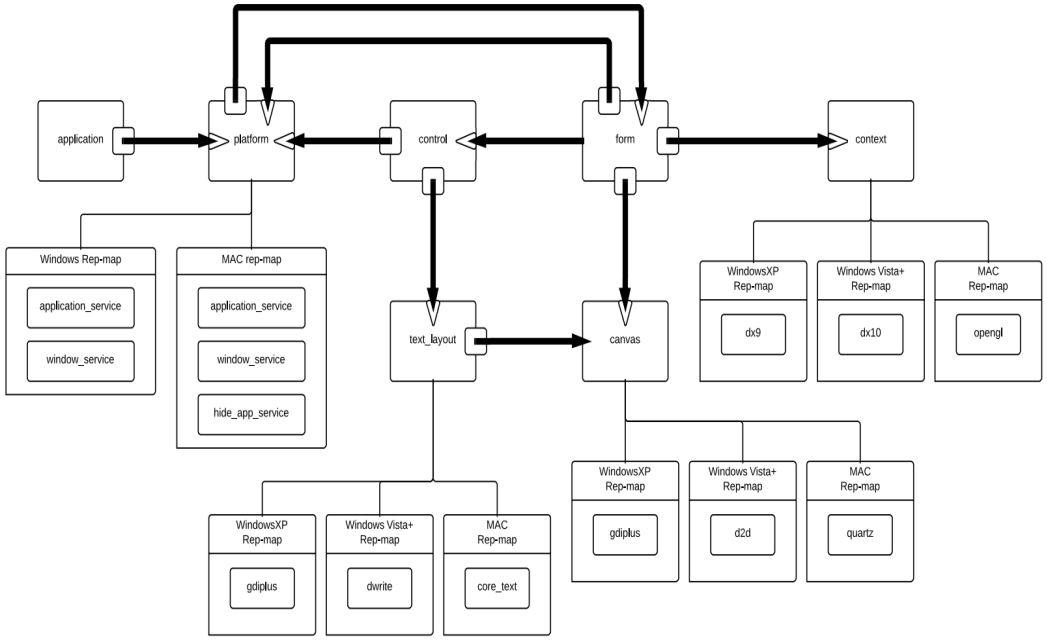


Fig. 3. FireMonkey Architecture

form 컴포넌트는 그림 6과 같이 구성되며 어플리케이션의 출력화면을 담당하는 특징이 있다.

```
Component form = {
 Ports { create; destroy; input_event paint }
}
```

Fig. 6. form component

```
Component context = {
 Ports { render; }

 Representation context_windows_xp = {
  System = context_windows_xp_system = {
   Component dx9 = {
    Ports { render; }
   }
  }
 }

 Representation context_windows_vista = {
  System = context_windows_vista_system = {
   Component dx10 = {
    Ports { render; }
   }
  }
 }

 Representation context_mac = {
  System = context_mac_system = {
   Component opengl = {
    Ports { render; }
   }
  }
 }

 Bindings {dx9.render to context.render}
 Bindings {dx10.render to context.render}
 Bindings {opengl.render to context.render}

}
```

Fig. 7. context component

context 컴포넌트는 그림 7과 같이 구성된다. context 컴포넌트는 3D 그래픽을 담당하는 컴포넌트로써, 다양한 3D 표현을 지원하고 있다.

canvas 컴포넌트는 그림 8과 같이 구성된다. 해당 컴포넌트는 context 컴포넌트와 비슷하게 2D 그래픽을 담당하는 컴포넌트로써, 다양한 2D 표현을 지원해 준다.

platform 컴포넌트는 그림 9와 같이 구성된다. platform 컴포넌트는 플랫폼에 종속적인 기능을 다루고 있는 여러 컴포넌트들을 가지고 있다. 또한 Control, Form 컴포넌트들이 플랫폼에 종속적인 기능을 사용하기 위해서는 이 컴포넌트를 통해 다른 컴포넌트들에 접근해야 한다.

```
Component canvas = {
 Ports { paint; }

 Representation canvas_windows_xp = {
  System = canvas_windows_xp_system = {
   Component gdiplus = {
    Ports { paint; }
   }
  }
 }

 Representation canvas_windows_vista = {
  System = canvas_windows_vista_system = {
   Component d2d = {
    Ports { paint; }
   }
  }
 }

 Representation canvas_mac = {
  System = canvas_mac_system = {
   Component quartz = {
    Ports { paint; }
   }
  }
 }

 Bindings {gdiplus.paint to canvas.paint}
 Bindings {d2d.paint to canvas.paint}
 Bindings {quartz.paint to canvas.paint}

}
```

Fig. 8. canvas component

그림 10의 text_layout 컴포넌트는 전반적으로 표현되는 모든 텍스트들을 관리하며 대표적로 텍스트의 크기 측정, 포맷 및 출력을 담당한다.

```
Component platform = {

 Representation platform_windows = {
  System platform_windows_system = {
   Component application_service = {
    Ports { run; terminate; handle_message; }
   }

   Component windows_service = {
    Ports { create; destroy; };
   }
  }
 }

 Representation platform_mac = {
  System platform_mac_system = {
   Component application_service = {
    Ports { run; terminate; handle_message; }
   }
```

```
   Component windows_service = {
    Ports { create; destroy; }
   }

   Component hide_app_service = {
    Ports { hide; }
   }
  }
 }

 Ports {input; application_intf; window_intf; hide_app;};

 Bindings {
  application_service.run to application_intf;
  application_service.terminate to application_intf;
  application_service.handle_message to input;

  window_service.create to window_intf;
  window_service.destory to window_intf;

  hide_app_service.hide to hide_app;
 }
}
```

Fig. 9. platform component

```
Component text_layout {
  Ports { render; }

  Representation text_layout_windows_xp = {
    System = text_layout_windows_xp_system = {
      Component gdiplus = {
        Ports { render; }
      }
    }
  }

  Representation text_layout_windows_vista = {
    System = text_layout_windows_vista_system = {
      Component dwrite = {
        Ports { render; }
      }
    }
  }

  Representation text_layout_mac = {
    System = text_layout_mac_system = {
      Component core_text = {
        Ports { render; }
      }
    }
  }

  Bindings {gdiplus.render to text_layout.render}
  Bindings {dwrite.render to text_layout.render}
  Bindings {core_text.render to text_layout.render}

}
```

Fig. 10. text_layout component

5. 멀티 플랫폼 지원 방안

소프트웨어 개발 시 멀티 플랫폼을 지원하기 위해서는 비즈니스 로직의 중요성 보다는 플랫폼과의 인터페이스나 전체적인 소프트웨어의 형상이 저 중요하게 인식되고 있다. 또한 플랫폼 별로 표현되는 방식과 구동원리가 다르기 때문에 UI가 중요한 요소 중의 하나가 될 수 있다. 따라서 멀티 플랫폼을 지원하는 소프트웨어 개발 툴은 다양한 UI를 플랫폼 별로 관리 하여 다양한 플랫폼에 탑재 가능한 소프트웨어를 개발할 수 있는 아키텍처가 필요로 하게 된다.

FireMonkey 아키텍처를 Acme를 통해 표현한 결과 멀티 플랫폼을 지원하기 위한 아키텍처는 어떻게 표현돼야 하는지 알 수 있었다.

비즈니스 로직의 경우 개발자가 선호하는 프로그래밍 언어를 사용하여 개발하면 되기 때문에 큰 문제가 존재하지 않지만, 플랫폼의 독립적인 특성을 극복하기 위해서는 다양한 플랫폼을 대상으로 UI 관련 데이터들을 끌어와 사용할 있도록 지원해야 한다. 그림 2의 platform 컴포넌트는 이러한 필요성에 따라 도출된 것이다. 또한 부가적으로 2D와 3D 그래픽, 텍스트를 처리하기 위해 그림 2에서 사용된 canvas 컴포넌트, context 컴포넌트, text_layout 컴포넌트와 같은 컴포넌트들이 필요로 하게 된다.

6. 결론

본 연구에서는 멀티 플랫폼의 소프트웨어 개발을 지원하는 FireMonkey 프레임워크를 Acme를 통해 아키텍처로 표현해보았다. 총 7개의 컴포넌트가 존재했고, 각각의 컴포넌트는 멀티 플랫폼을 지원하기위한 관계를 가지고 있었으며, 플랫폼 종속적인 내용은 platform 컴포넌트를 통해 일괄적으로 처리해주는 것을 확인할 수 있었다. 현재 데스크톱 플랫폼 기준으로 Windows 계열과 Mac만을 제공하고 있지만 추후 모바일 플랫폼을 지원하기 위해 Android와 iOS의 종속적인 정보들을 platform 컴포넌트를 기준으로 추가하여 기능을 확장하고자 한다. FireMonkey 아키텍처는 Acme로 표현되었기 때문에 다른 ADL로 변경이 용이한 점을 활용하여 다양한 분석을 실시할 수 있다.

참 고 문 헌

[1] P. Clements, R. Kazman, M, Klein, “Evaluating Software Architecture: Methods and Case Studies”, Addision-Wesley, 2002.

[2] J. Bosch, “Design and Use of Software Architecture”, Addison-Wesley, 2000.

[3] Wojtczyk, Martin, and Alois Knoll. "A cross platform development workflow for C/C++ applications." Proceeding on Software Engineering Advances, 2008.

[4] Embarcadero Technologies, “FireMonkey”, http://www.emb arcadero-inf o.com/firemonkey, 2012.

[5] D. Garlan, R. Monroe, D. Wile, “Acme: an architecture description interchange language”, Proceeding on conference of the Centre for Advanced Studies on Collaborative research, 1997.

[6] A. Dragomir, H, Lichter, “Model-based Software Architecture Evloution and Evaluation”, Proceeding on conference of the Asia-Pacific Software Engineering, 2012.

[7] X. Cui, Y. Sun, S. Xiao, H. Mei, “Architecture Design for the Large-Sacle Software-Intensive Systems: A Decision-Oriented Approach and the Experience”, Proceeding on conference of the International Conference on Engineering of Complex Computer Systems, 2009.

[8] F. Buschmann, K. Henney, Douglas C. Schmidt, “Pattern-oriented Software Architecture: On Patterns and Pattern Languages”, John Wiley and Sons, 2007.

[9] M. Rong, C. Liu, G. Zhang, “Modeling Aspect-oriented Software Architecture Based on ACME”, Proceeding on conference of the International Conference on Computer Science & Education, 2011.

[10] Z. He, K. Ben, Z. Zhang, “Software Architectureal Reflection Mechanism for Runrime Adaptation”, Proceeding on conference of the International Conference for Young Computer Scientists, 2008.

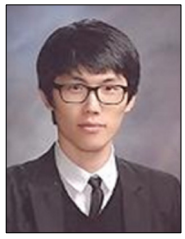
저 자 소 개

박 재 진(Jae-Jin Park) [정회원]

- 2012년 8월 : 충북대학교 컴퓨터공학부 (학사)
- 2012년 9월 ~ 현재 : 충북대학교 컴퓨터과학과 (석사과정)

<관심분야> : 소프트웨어 품질, 코드 리팩토링, 소프트웨어 아키텍처, 객체지향 모델링

고 재 철(Jae-Cheol Ko) [정회원]



- 1998년 2월 : 충북대학교 컴퓨터공학부 (학사)
- 2010년 2월 ~ 현재 : 충북대학교 컴퓨터과학과(석사과정)

<관심분야> : 소프트웨어 아키텍처, 소프트웨어 품질, 객체지향 프로그래밍

홍 장 의(Jang-Eui Hong) [정회원]

- 2001년 2월 : KAIST 전산학과 (공학박사)
- 2001년 2월 ~ 2002년 9월 : 국방과학연구소 선임연구원
- 2002년 10월 ~ 2004년 8월 : (주) 솔루션링크 기술연구소장
- 2004년 9월 ~ 현재 : 충북대학교 소프트웨어학과 교수

<관심분야> : 소프트웨어 품질공학, 모델기반 검증 분석, 소프트웨어 아키텍처, 소프트웨어 프로세스 개선