API 정보 저장소를 활용한 동적 재구성 지원 시스템의 설계

윤 석진*, 김 선자*, 김 현수**
*한국전자통신연구원, **충남대학교

Design of The Dynamic Binding Systems using API Information Repository

Yoon Seok-Jin, Kim Sun-Ja°, Kim Hyeon-Soo**
°Electronics Telecommunication Research Institute, **Chung-Nam University
E-mail : sjyoon@etri.re.kr, sunjakim@etri.re.kr, hskim401@cnu.ac.kr

요 약

오늘날 운영체계와 미들웨어는 수많은 API를 제공하고 있다. 최종 사용자들이 사용하는 응용 프로그램들은 이러한 API를 활용하여 개발되어지고 있다. 기존의 문서와 같은 형태의 API에 대한 기술은 기계가 자동적으로 처리하기 힘들며 개발자 입장에서도 API를 이해하기 위해서 책을 직접 읽고 이해해야 한다는 단점이 있다. 이를 극복하기 위하여 docgen과 같은 도구들이 있으나 결국은 개발자 관점에서는 API를 직접 숙지하여야 하며 응용 프로그램 상에서 직접 운영체계에서 제공하는 API 호출 부분을 작성하여 개발하여야 한다는 부분은 동일하다. 또한 서로 다른 다양한 운영체계에서는 형식은 다르지만 유사한 기능을 제공을 하는 API들이 있으나 개발자는 특정 운영체계의 API에 맞추어서 각각의 운용체계에 맞는 응용 프로그램을 개발하여야 하는 문제점이 있다.

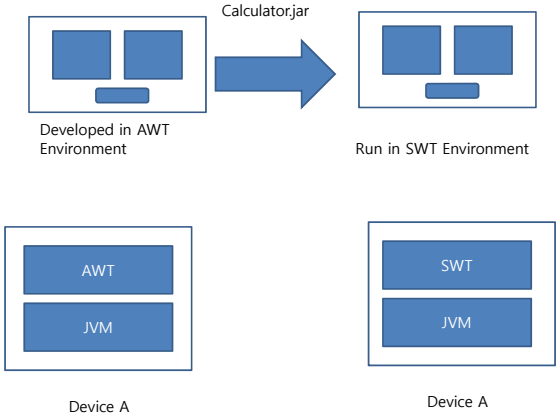
본 연구에서는 API에 대한 정보에 대한 규격을 정의하고 각각의 API의 기능 및 특성에 대하여 메타 기술언어를 사용하여 기술하여 저장소에 저장해두고 실행시에 이러한 API정보를 사용하여 응용 프로그램과 동적으로 바인딩시켜서 실행시키는 체계에 대해서 논의한다. 응용 프로그램에서 사용하려고 하는 API와 운영체계에서 제공하는 API를 동적으로 결합시키는 엔진에서는 운영체계별 API의 차이에 의한 부분을 상쇄시키는 역할을 담당한다. 이러한 체계를 활용하면 동적으로 재구성 가능한 응용을 개발하는데 있어서 하부 시스템으로 활용될 수 있다. 향후 표준 API를 기술하는데 있어서도 본 연구에서 제시하는 메타 방식의 API 기술표현을 활용하면 보다 정확한 표준 규약 준수 여부를 확인할 수 있는 방법을 제공할 수 있다.

1. 서론

오늘날 운영체계와 미들웨어는 응용 프로그램에서 사용할 수 있는 수많은 API를 제공하고 있다. 최종적으로 사용자들이 사용하는 응용 프로그램들은 이러한 API를 활용하여 개발된다. 일반적인 개발 방법은 개발자가 API를 이해하기 위해서 관련된 서적 및 문서를 직접 읽고 이해해야 한

후에 최종 사용자에게 서비스를 제공하기 위한 알고리즘을 운영체계 및 미들웨어가 제공하는 API를 호출하는 프로그램을 작성하는 과정을 거친다. 이러한 과정은 그동안 자동화되기 힘들었으며 새로운 운영체계나 업그레이드된 미들웨어가 나올 경우 다시 재 숙지하여 재개발해야 하는 경우가 많았다.

개발자가 이해하는 과정을 돕기 위하여 docgen과 같은 API 문서 생성 도구 및 이해지원 도구들이 있으나 최종적으로는 개발자 입장에서 API를 직접 숙지하여야 하며 응용 프로그램 상에서 직접 운영체계에서 제공하는 API 호출 부분을 작성하여 개발하여야 한다는 부분은 동일하다. 또한 서로 다른 다양한 운영체계에서는 형식은 다르지만 유사한 기능을 제공을 하는 API들이 있으나 개발자는 특정 운영체계의 API에 맞추어서 각각의 운용체계에 맞는 응용 프로그램을 개발하여야 하는 문제점들이 여전히 존재한다.



<그림 1> 서로 다른 환경에서의 실행

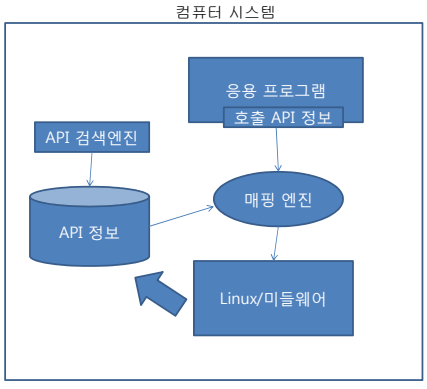
<그림 1>은 본 연구에서 해결하려고 하는 문제에 대한 예시이다. Calculator라고 하는 프로그램은 장치 A를 위해 Java의 AWT 라이브러리를 사용하여 UI를 개발하였다. 하지만 모바일 환경과 같은 다종의 단말이 존재하는 경우에 때에 따라서 장치 B와 같이 AWT 라이브러리 대신에 SWT라이브러리가 탑재되는 경우가 종종 발생한다. 이러한 경우 개발자는 Calculator라는 서비스가 제공하는 기능을 그대로 유지하지만 SWT라는 환경에서 동작하게 하기위한 또하나의 CalculatorForSWT 버전을 개발하여야 한다.

본 연구에서는 이와 같은 문제점들을 해결하기 위하여 이미 개발된 응용 프로그램이 버전이 다른 운영체제나 미들웨어에서도 동작을 보장하기 위해서 운영체제 및 미들웨어에서 제공하는 API에 대한 정보에 대한 저장 규격을 정의하고 각각의 API의 기능 및 특성에 대하여 메타적인 기술언어를 사용하여 표현하여 정보 저장소에 저장한다. 해당 응용 프로그램의 실행시에 이러한 API정보를 사용하여 응용 프로그램에서 필요로 하는 API와 운영체계에서 제공하는 API를 동적으로 바인딩시켜서 실행시키는 체계에 대해서 논의한다.

..

2. 본론

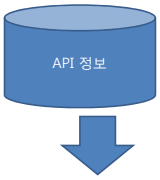
본 연구에서 제시하는 API간의 맞춤형 적응 시스템은 API 정보의 구성 방법과 이를 탑재한 플랫폼의 구성 방법과 API의 정보를 조사할 수 있는 API 검색엔진, 응용 프로그램을 실행할 경우 API 정보를 활용하여 유사한 API를 검색하여 매핑시키는 매핑 엔진으로 구성된다.



<그림 2>. API 정보를 포함하는 플랫폼의 구성도

<그림 2>는 API 정보를 포함하는 플랫폼의 구성을 나타낸다. 플랫폼은 일반적으로 운영체계와 실행에 필요한 미들웨어, 응용 프로그램으로 구성되며, 운영체계와 미들웨어는 응용 프로그램이 사용할 수 있도록 응용 프로그래밍 인터페이스(API)를 제공한다. 응용 프로그램은 이러한 API들을 조합하여 특정한 기능을 수행하기 위한 프로그램을 구성하게 된다. 본 연구에서는 운영체계와 미들웨어가 제공하는 API에 대한 정보를 별도의 데이터베이스로 구성하여 플랫폼에 같이 탑재를 시킨다.

운영체계와 미들웨어와 같이 탑재된 API정보 데이터베이스는 API 검색엔진을 이용하여 개발자나 사용자가 필요할 경우 플랫폼에서 제공하는 API들을 조사할 수 있다. 또한 응용 프로그램은 자신이 사용하는 API에 대한 정보를 별도로 관리하여 응용 프로그램이 실행될 경우 매핑엔진이 응용 프로그램이 사용하는 API와 실제 운용체계에서 제공되는 API를 API정보 데이터베이스를 통하여 검색하고 이 정보를 이용하여 유사한 API인 경우 대신 호출을 해주는 매핑엔진을 같이 탑재할 수 있다.

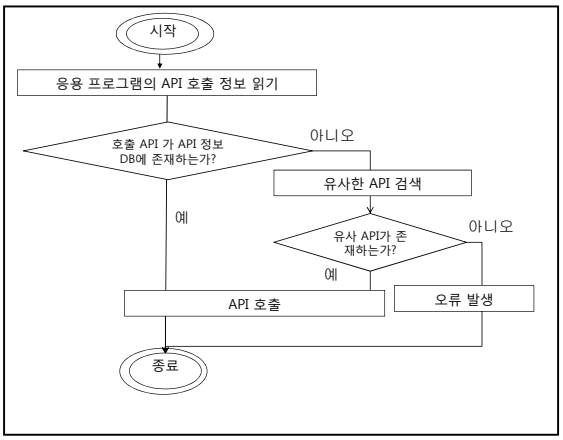


API 이름	클래스	라이브러리	파라미터	의미	기타 정보
drawpixel	Graphic:gtk	Libgtk.so	X, y, color	Draw pixel	TBD
rectangle	Graphic:gtk	Libgtk.so	Topx, uppery, width, height	Draw rectangle	TBD

<그림 3> API 정보의 구성 예

<그림 3>는 API 정보 데이터베이스에 저장되는 API 정보의 한 예이다. API의 호출 이름과 API가 속한 클래스(계급, 분류) 정보를 갖고 있다.

또한 API가 속한 물리적인 파일 이름 정보, API에 전달하여야할 파라미터 정보를 갖고 있다. 그리고 해당 API가 의미하는 의미 정보를 갖는다. 의미 정보는 자연어로 기술될 수도 있고 필요한 경우 시멘틱 웹과 같은 형태의 기계 장치가 인식할 수 있는 형태로 기술되어져 매핑엔진이 쉽게 검색 및 인식할 수 있도록 한다. 이외에 필요한 성능 정보, 제약조건등의 기타정보를 갖고 있다. 이러한 API 정보의 구성은 경우에 따라 변경될 수 있으나 개별 API 별로 작성되어서 API 정보 데이터베이스에 저장된다.



<그림 4>. 매핑을 위한 검색 과정

<그림 4>는 응용 프로그램을 수행할 경우에 매핑 엔진을 통해서 유사 API와 결합하는 과정을 설명한다. 먼저 응용 프로그램에서 호출하는 API가 API 정보 데이터베이스에 존재하는지 검색한다. 있을 경우에는 바로 API를 호출하고 없을 경우에는 유사한 API가 있는지 검색한다. 유사한 API가 있을 경우 호출 API와 유사 API간의 매핑 과정을 수행하고 매핑을 할 수 없거나 유사한 API가 존재하지 않을 경우에는 오류를 발생시킨다.

매핑 과정이 끝나면 이 정보를 토대로 하여 바인딩하기 위한 랩퍼 클래스를 생성한다. 랩퍼

클래스는 컴파일되어 클래스 파일을 생성하고 이러한 클래스 파일은 실행시에 바인딩되도록 한다. 이러한 과정은 실행 전에 프로그램을 분석하여 수행할 수도 있고 Java 언어의 특성을 이용하면 실행 중간에 컴파일하여 동적 연결을 수행하게 할 수도 있다.

.

3. 결론

본 연구에서 제시하는 응용 프로그램에서 사용하려고 하는 API와 운영체계에서 제공하는 API를 동적으로 결합시키는 엔진에서는 운영체계별 API의 차이에 의한 부분을 상쇄시키는 역할을 담당한다. 이러한 체계를 활용하면 동적으로 재구성 가능한 응용을 개발하는데 있어서 하부 시스템으로 활용될 수 있다

대부분의 경우는 유사한 API를 검색할 필요가 없이 운영체계와 미들웨어에서 제공하는 API와 바로 연결되는 경우이겠지만 간혹 버전의 차이 등으로 인해 수행되지 못할 경우 본 연구에서 제시하는 방법의 경우 유사한 API를 검색하여 수행시킴으로서 응용 프로그램의 수행시킬 수 있는 확률을 높여줄 수 있다. 수행한 결과의 이상이 없음을 보장하기 위하여 추가적으로 API 정보 데이터베이스에 보장하기 위한 정보를 더 추가할 수 있다.

이러한 기능을 운영체계에서 기본으로 제공하게 되면 응용 프로그램의 호환성을 보다 더 높여줄 수 있으며 시스템이 응용 프로그램의 의도를 파악해서 스스로 적응하게 할 경우 하나의 방법으로 제공될 수 있다.

향후 API를 기술하는 표준 방법에 대한 연구에 있어서도 본 연구에서 제시하는 메타 방식의 API 바인딩 방안을 활용하면 보다 정확한 표준 규약 준수 여부를 확인할 수 있는 체계를 제공할 수 있다.

[참고문헌]

[1] Ontology and Application to Improve Dynamic Bindings in Mobile Distributed Systems, Ben Falchuk, Dave Marples, WICON'06, The 2nd Annual International Wireless Internet Conference, August, 2006,

[2] QoS의 동적 적응이 가능한 웹 서비스 매치메이킹, 강필석외 4인, 제27회 한국정보처리학회 춘계학술발표대회 논문집 제14권 제1호, 2007. 5

[3] A Framework and Ontology for Dynamic Web Services Selection, E. Michael Maximilien & Munindar P. Singh, IEEE INTERNET COMPUTING, SEPTEMBER . OCTOBER 2004