다양한 버전의 미들웨어 생성을 위한 인터페이스 보정 도구의 설계

장 기 영 , 송 영 재
경희대학교 컴퓨터공학과 소프트웨어공학 연구실
forthetop@lycos.co.kr

The Design of Interface Refining Tool for Creating Diverse Versions of Middleware

Jang, Ki-Young , Song, Young-Jea
KyungHee University, Department of Electronics and Information, Computer Engineering, Software Engineering Lab.

요 약

임베디드 시스템의 중요성이 대두됨에 따라 미들웨어를 개발하는 데에 있어서 다양한 플랫폼에 맞도록 여러 버전을 빠르게 생성할 필요성이 대두되었다. 그래서 객체지향과 CBD를 기반으로 한 여러 연구들이 있었으나 대부분 응용 분야에 초점이 맞춰져, 미들웨어의 개발에는 현실적으로 전통적인 개발자의 수작업이 주를 이루었다. 본 논문에서는 기존 미들웨어의 새로운 버전일 필요할 때에 기존 모듈과 새로 개발된 모듈의 여러 조립과정을 간단하게 처리할 수 있도록 하는 도구를 사용할 것을 제안한다. 이 도구는 모듈간의 조립에 필요한 정보를 축적하는 등록과정과 실제 자동으로 모듈의 조립을 수행하는 인터페이스 보정의, 2단계로 구성된다. 이러한 도구를 통해 개발자는 미들웨어의 각 기능을 개발하는 데에만 집중하여 생산성을 높일 수 있다.

1. 서론

컴포넌트 기반 개발 방법은 오늘날의 소프트웨어 공학 분야에서 빠지지 않는 기본 개념으로 자리하고 있다. 많은 새로운 연구들이 CBD를 기반으로 한 새로운 결과들을 내어놓고 있다. 또한 실제 개발 현장에서도 컴포넌트 기반 개발을 지원하는 다양한 도구들이 제작되어 CBD를 성공적으로 프로젝트에 적용시킨 사례들도 많이 존재한다.

그러나 미들웨어의 개발에 있어서는 CBD의 개념들과 장점이 충분히 적용되고 있지 못하다. 특히 기존 미들웨어에 기능을 추가하거나 삭제 또는 변경한 새로운 버전을 만들어내는 경우, 해당 모듈의 구현뿐만 아니라 전체 모듈들의 조립도 개발자가 직접 하는 것이 보통이다. 이 경우 자연스럽게 개발자가 여러 모듈에 걸쳐 코드를 변경하는 작업을 하는 것이 당연하였고, 따라서 생산성이 떨어지며 안정성도 떨어지게 된다. 그런데 미들웨어는 전혀 새로운 것을 개발하는 요구보다는 기존 미들웨어를 다른 플랫폼에 이식하는 요구가 더 높기 때문에 기존 미들웨어의 새로운 버전을 이전보다 빠르게 생성해야 할 필요성이 대두되었다.

그러나 이전까지의 CBD나 객체지향에 관련된 연구들은 이러한 미들웨어의 개발에 있어서는 적합하지가 않았다. 본 논문에서는 그 중에서도 대표적인 몇 가지 연구들을 기반으로 장점을 취하고 단점을 보완하는 방법을 모색했다. 또한 컴포넌트 기반 개념 중에서 모듈의 독립성에 의한 장점을 적용하고, 또한 이식성과 생산성을 높여서 다양한 플랫폼에 더 빠르고 쉽게 적용할 수 있도록 하는 모듈들의 조립 구조와 그 조립과정을 자동화하는 도구에 대한 것을 설명한다.

2. 기존 미들웨어 설계의 특징들

기존 연구들에서는 미들웨어의 개발에 CBD를 적

극적으로 적용한 예를 찾기가 매우 힘들다. 대부분의 연구들은 응용 소프트웨어나 비즈니스 솔루션을 대상으로 한 것들이다. 따라서 여기에서는 기존의 연구들을 살펴서 보다 더 많이 미들웨어에 관련된 설계구조 및 프레임워크를 살펴본다. 연구들은 크게 두 가지 분류로 나누어 접근할 수 있다.

2.1. CORBA를 기반으로 한 설계들의 특징.

한 가지는 CORBA를 기반으로 하는 연구들이다. 이러한 연구들에서는 미들웨어 표준 중에서 가장 널리 퍼지고 사용되고 있는 CORBA의 장점들을 그대로 이어받고자 한다. CORBA는 말 그대로 '표준' 이기 때문에 CORBA 기반을 이어가는 것으로 미들웨어 자체가 쉽게 사용될 가능성이 커지는 효과를 볼 수 있다. 또한 더 나아가 CORBA의 특정 부분의 내부 구조를 조금 변경하여 성능적인 향상을 이끌어내거나, 적용방법을 달리하여 상이한 플랫폼에서 CORBA의 장점을 충분히 이끌어내기 위한 내용들이 주류를 이룬다.

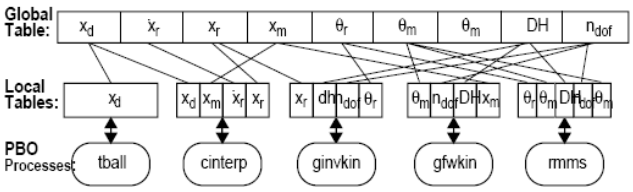
그러나 이러한 방법들은 자원 제약이 심한 플랫폼에 CORBA를 이용하려고 할 때에 해당 플랫폼에 맞게 적절히 기능을 제한한 새로운 버전을 만들어야 할 필요가 있다. 그리고 CORBA는 그 자체가 분산 객체의 통신을 위한 것이기 때문에 미들웨어를 이루는 각 모듈들을 CORBA의 형식으로 조립 및 구성하는 것은 적합하지 않다. 왜냐하면, 미들웨어의 모듈들이 네트워크에 분산되어 있는 경우는 없기 때문이다. 즉, CORBA의 분산 객체를 위한 주요 기능과 구조는 미들웨어를 구성하기 위한 설계에서는 불필요한 요소이다.

2.2. CORBA 기반이 아닌 설계들의 특징.

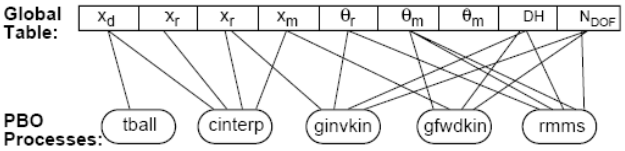
CORBA를 기반으로 하지 않는 다른 한 가지는 기존의 널리 알려진 표준이나 프레임워크와는 다른 새로운 설계 구조를 제시하는 연구들이다. 이러한 것들은 각각이 성취하기 위한 특별한 목적를 가지고 있으며, 그러한 목적에 집중된 새로운 설계 구조를 제안하고 있다. 따라서 그 종류가 매우 많고, 다양한 상황에 적용하기에는 오히려 CORBA 기반의 설계 구조보다 적절하지 않다.

그들 중 본 논문에서 주로 참조한 PBO(Port Based Object)모델을 예로 들어보자.

PBO는 Port Based Object의 약자로 각 객체가 Port의 역할을 하는 공통의 자료구조를 두고, 그곳을 통해서 메시지에 해당하는 데이터를 주고받도록 하는 것이다. 그림 1과 2는 그것을 표현한 것이다.



(그림 1) PBO의 데이터 테이블 구조 (선점형)



(그림 2) PBO의 데이터 테이블 구조 (비선점형)

그림 1과 그림 2는 둘 모두 PBO의 각 객체들이 데이터를 주고받는 형태를 표현한 것이다. 그림에서 알 수 있듯이 각 PBO 객체들은 일반적인 의미의 모듈이라기보다는 하나의 Process 또는 Thread 단위로 봐야 한다. 따라서 Process간의 데이터 교환 방법이 선점형과 비선점형의 형태에 따라 달라지므로 그림처럼, 두 가지 결과가 나올 수 있다.

공통적인 것은 Process간에 비동기 통신이 이루어지고 있는 것이며, Global Table 내의 값들이 각기 Queue의 구조로 복수의 메시지를 순서대로 저장할 필요가 있다는 것이다. 또한 모든 객체가 Process로 동작하기 위해 공통 인터페이스를 상속해야 하는데, C의 매크로를 활용한 코드 생성기법을 적용하여 객체지향 언어에서의 인터페이스 상속 효과를 구현하였다.

3. 제안된 인터페이스 보정 도구의 요점 정리.

이상의 내용을 바탕으로 새롭게 제안될 설계 구조와 도구의 필요조건을 다음과 같이 정리할 수 있다.

우선 프레임워크의 구조가 매우 간단해야 한다. 일반적인 모듈간의 함수 호출을 전혀 다른 메시지 전달 방식으로 바꾸는 것은 좋지 않다. 가능한 유지하면서 그 사이의 호출이나 연결을 자동화 시키는 것이 더 중요하다. 또 구조가 복잡하면, 미들웨어 자체의 이식성에 좋지 않은 영향을 끼칠뿐더러 미들웨어가 다른 미들웨어 위에 존재하는 형태가 될 수도 있다. CORBA기반의 예에서 보았듯이 불필요한 기능들이 없어야 플랫폼 이식성이 높아진다.

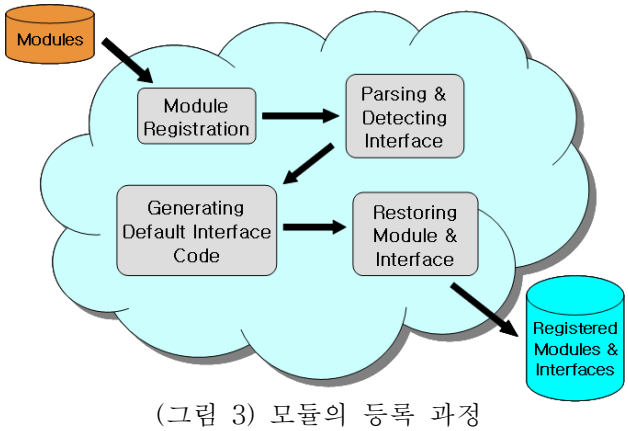
또한 미들웨어를 구성하는 각 컴포넌트의 독립성을 높여서, 컴포넌트의 교체가 쉽고 간단하여야 한다. 이것은 미들웨어의 각 모듈들의 통신 구조가 정해진 규칙이나 형식을 따르도록 작성되거나, 개발도구를 이용하여 자동으로 생성될 때에 얻을 수 있는 장점이다.

이러한 설계 구조와 도구를 만들기 위해서 C언어로 작성된 모듈의 소스 코드를 직접 읽어서 인터페이스를 추출해 내기로 하였다. 이러한 방법은 매우 제한된 조건에서만 효과적인 방법이다. 호출하는 쪽과 호출되는 쪽의 호출양식이 정확히 맞지 않으면 자동으로 맞추는 것은 소스 코드의 해석만으로는 거의 불가능하기 때문이다. 기존의 연구들은 XML을 이용하여 컴포넌트의 인터페이스를 명세하고 XML을 이용해 서로 다른 호출 사이의 변환을 꾀하였다. 그러나 이러한 방법도 한계가 있으며, 또한 여기에서 말하는 도구에는 필요하지 않다.

그 이유는 전혀 새로운 미들웨어를 개발해야하는 요구보다 기존의 미들웨어를 약간 변형한 다른 버전의 미들웨어를 빨리 개발해야하는 요구가 훨씬 높기 때문이다. 기존의 미들웨어의 기능을 약간 변형한다면 함수 내부 구현의 변화, 새로운 함수의 추가 및 기존 함수의 제거 등이 주 변경사항으로, 여기에서는 이미 함수들의 호출 양식이 정해져 있으며. 어떤 모듈에서 어떤 모듈로 호출하는지조차 정해져 있다고 봐야한다. 따라서 함수의 호출 양식이 거의 바뀔 일이 없고 소스 코드의 분석으로 얻은 정보로도 모듈간의 호출을 매개하는 인터페이스를 생성하는 데에 무리가 없다.

3.1. 인터페이스 보정 도구의 등록 과정.

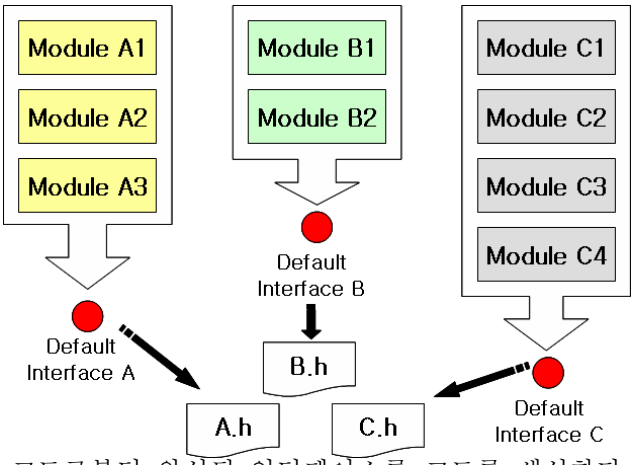
그림 3은 일반적인 C 모듈을 등록하는 과정을 설명한다. C 모듈들은 헤더 파일을 하나씩 가지고 있다. 등록 과정은 먼저 등록할 C 모듈의 소스 코드를 선택하고. 각 헤더파일을 읽어서 인터페이스 정보들을 추출해 낸다. 추출한 정보를 바탕으로 기본 인터페이스 코드를 생성해 낸다. 이 인터페이스 코드는 호출을 하는 쪽에서 항상 호출하는 코드로 기본 인터페이스의 코드를 바꿈으로써 모듈의 교체, 추가, 삭제를 구현한다. 마지막으로 생성된 기본 인터페이스 코드와 자동생성에 대한 부가정보를 원래의 모듈과 함께 등록된 모듈을 저장하기 위한 공간에 정리하여 저장한다.



(그림 3) 모듈의 등록 과정

3.2. 인터페이스 보정 도구의 조립 과정.

각 C 모듈을 모두 등록시킨 후에는 등록된 모듈을 조립하여 여러 버전의 미들웨어를 생성하는 단계에 들어간다. 생성하기 전에 우선 어떤 모듈들을 포함 시킬 것이며, 포함시키는 모듈이 여러 버전이 존재할 경우에 어떤 버전을 포함시킬지를 개발자가 선택해야 한다. 선택한 정보에 따라 기본 인터페이스 코드로부터 완성된 인터페이스를 코드를 생성한다.



(그림 4) 인터페이스 코드 보정

그림 4에는 모듈 A, B, C가 각각 3개 2개 4개의 다른 버전으로 작성된 상태에서 기본 인터페이스 A, B, C가 작성되는 모습을 보여준다. 각 기본 인터페이스는 해당 모듈의 여러 버전들에 있는 인터페이스를 모두 포함한다. 예를 들어, 기본 인터페이스 C.h 파일은 모듈 C의 4개 버전의 모든 함수, 매크로, 데이터 선언 등을 포함하는 것이다.

따라서 어떠한 버전을 포함시킬지를 선택한다면, 선택한 버전의 모듈에서는 제공하지 않지만, 기본 인터페이스에는 선언되어 있는 함수, 매크로, 데이터들이 생겨난다. 그래서 이러한 요소들에 대한 접근

및 호출이 있을 때에는, 경고 메시지 등으로 개발자에게 알릴 수 있도록, 자동 생성된 코드로 들어가야 한다. 그리고 이러한 과정을 거친 완성된 인터페이스 코드가 각 모듈의 헤더 파일을 대신한다. 즉, 모듈의 접근이 있을 때, 인터페이스가 보정되어 대체된 헤더 파일이 원래 모듈의 헤더 파일을 대신한다는 뜻이다.

그렇기 때문에 이렇게 대체된 헤더파일은 원래 호출하는 측에서 참조하는 헤더파일과 이름이 같아야 한다. 이렇게 함으로써 호출하는 측의 코드는 변경하지 않아도 되는 장점이 있다. 즉 모듈간의 독립성이 높아진 효과를 얻는다. 또한 해당 모듈이 아예 제외되는 상황에서도 호출하는 측의 코드는 변하지 않고 여전히 해당 모듈의 헤더를 포함해야만 모듈간의 독립성을 지킬 수 있다. 이러한 경우에는 위에서 설명한 방법을 그대로 사용하여 실제 호출이 일어났을 경우에 해당 모듈이 아예 제외되었다는 메시지를 개발자에게 알리도록 자동 생성된 코드가 들어가야 한다.

4. 결론

미들웨어의 다양한 버전을 빠르게 생성하기 위해서는 정교한 메시지 교환 시스템이 필요한 것이 아니라 사용자가 간편히 모듈에 대한 설정을 하고 그것을 자동으로 적용시켜주는 도구가 더 필요하다. 이러한 도구는 미들웨어를 이루는 각 모듈에 대한 설정을 사용자로부터 입력받고 가능한 모듈 자체에 대한 변화는 주지 않으면서 모듈의 교체, 추가, 삭제가 이루어지도록 하는 것이 좋다.

이러한 목적을 이루기 위해 미들웨어의 각 모듈들을 등록하여 관리하는 기능을 제안하였다. 또한 등록된 모듈들의 헤더파일로부터 얻어낸 인터페이스 정보들이 충분히 유용함을 밝혔고, 그러한 정보들로부터 만들어진 인터페이스 코드를 호출의 중간에 삽입함으로써 기존 모듈에는 변경이 없이 조립할 수 있는 방법을 제안하였다.

참 고 문 헌

[1] David B. Stewart "Designing Software Components for Real-Time Applications, 2001 Embedded Systems Conference San Francisco, CA, April 2001.

[2] Romain Rouvoy, Philippe Merle. "Towards a model-driven approach to build component-based adaptable middleware", Proceedings of 3rd workshop on Adaptive and Reflective Middleware 2004. pp.195-200.

[3] Irfan Pyarali, Carlos O'Ryan, Douglas Schmidt, Nanbor Wang, Vishal Kachroo, "Using Principle Patterns to Optimize Real-Time ORBs", IEEE Concurrency, Vol 8, Issue 1, January 2000, pp.16-25.

[4] Joan Krone, "Multiple implementations for component based software using Java interfaces", Journal of Computing sciences in Colleges, Vol 19, Issue 1, October 2003, pp.30-38.

[5] Avraam Chimaris, George A. Papadopoulos, "Control-Driven Coordination Based Assembling of Components", Computer Software and Applications Conference, 2002. COMPSAC, 26th Annual International, August, 2002, pp.572-577.

[6] Frederic Doucet, Sandeep Shukla, Rajesh Gupta, Masato Otsuka, "An Environment for Dynamic Component Composition for Efficient Co-Design", Automation and Test in Europe Conference and Exhibition, 2002. March 2002, pp.736-743.

[7] Alejandra Cechich, Manuel Prieto, "Comparing Visual Component Compositiion Environments", Computer Science Society, 2002. SCCC 2002. Proceedings. November 2002, pp.217-225.

[8] A.David McKinnon, Kevin E. Dorow, Tarana R. Damania, Olav Haugan, Wesley E. Lawrence, David E. Bakken, John C. Shovic, "A Configurable Middleware Framework with Multiple Quality of Service Properties for Small Embedded Systems", Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on 16-18, April 2004, pp.197-204.

[9] Zonghua Gu, Sharath Kodase, Shige Wang, Kang G. Shin "A Model-Based Approach to System-Level Dependency and Real-Time Analysis of Embedded Software", Real-Time and Embedded Technology and Applications Symposium, 2003. Proceddings. The 9th IEEE 27-30, May 2003, pp.78-85.