사용자 DLL의 동적 플러그인을 위한 메시지 객체의 인터페이스 설계 방법

심준용○ 이용헌 조규태 김세환
엘아이지넥스원(주)
jyshim79@lignex1.com, yhlee80@lignex1.com kyutcho@lignex1.com saehwan.kim@lignex1.com

An Interface Design Method of the Message Object for a Dynamic Plug-in Dynamic Linked Library

Junyong Shim○ Yongheon Lee Kyutae Cho Saehwan Kim
LIG Nex1 Co., Ltd.

개발 시스템을 성공적으로 구축하기 위해서는 시스템 요구사항에 적합하고 다양한 품질 속성을 만족시킬 수 있는 소프트웨어 아키텍처 설계와 공통의 실행 환경을 제공하는 프레임워크 개발이 필요하며, 개발 시스템은 사용자 요구사항 변경에 대한 용이성, 구성 요소의 신뢰성 및 재사용성을 높일 수 있는 구조를 제공해야 한다. 국방 분야에서도 무기체계 또는 비 무기체계 개발 시 이러한 품질을 보장할 수 있는 기반 기술을 요구하고 있으며, 특히 모델링 및 시뮬레이션을 기반으로 하는 시스템 구축 시 High Level Architecture(HLA)와 같은 기술 표준 적용을 권고하고 있다. 당사는 HLA 기반의 M&S 프레임워크를 개발하여 다수의 시뮬레이션 모의기 개발을 위한 공통의 개발 환경을 마련했다. 한편, M&S 프레임워크는 구성 요소들을 코드 수준에서 재사용하기 때문에 시뮬레이션 규모가 크거나 다수의 개발자가 참여할 경우 모듈 간 인터페이스 및 통합시험 과정이 복잡해지고, 프레임워크 수준에서 요구사항 변경 요청이 있을 경우 적용된 모든 시뮬레이터의 수정이 불가피 하다.

플러그인 플레이 방식의 아키텍처는 이러한 문제를 해결할 수 있는 구조를 제시하며, 특히 프레임워크의 구성 모듈을 DLL(Dynamic Linked Library)로 구현하고, 명시적 연결(Explicit Linking)을 사용함으로써 기능의 동적 재구성이 가능하다. 따라서 모듈의 조립 또는 분해만으로 변경된 요구사항의 수정이 용이하고, 시스템 개발에 참여하는 구성원 간 의존성을 줄일 수 있다. 하지만 DLL의 명시적 연결은 DLL을 개발하는 시점에서 프레임워크와 상호작용하는 인터페이스 구현에 접근할 수 없다는 문제를 갖고 있다. 즉, 프레임워크는 EXPORT 함수를 통해 DLL 접근이 용이하지만 DLL의 경우 프레임워크 내 접근 클래스에 대한 객체의 참조를 가져와야 하기 때문에 프레임워크와 밀접한 의존성이 발생하게 된다.

본 논문은 DLL 구현 시 프레임워크의 상호작용 클래스에 대한 널(Null) 객체를 정의하여 프레임워크와 독립적인 구현 환경을 제공할 수 있는 메시지 객체의 인터페이스 설계 방법을 제시한다. 제안하는 방법은 프레임워크에 플러그인하는 DLL이 상호작용 하는 DLL의 라우팅 정보와 관계없이 데이터를 교환할 수 있도록 데이터 중심 교환방식을 사용하는 메시지 프로토콜을 사용하며, 교환 메시지는 XML 형식으로 작성하여 메시지 해석기(Parser)를 통해 DLL 간 라우팅 테이블을 자동으로 생성할 수 있도록 설계했다. 특히, 플러그인 수행 후 널(Null) 객체를 프레임워크로부터 생성된 실제 객체로 동적바인딩 시켜줌으로써 완전한 플러그인을 기능을 구현했다.

일반적으로 DLL을 사용하는 방법은 정적 링크와 동적 링크로 나뉘는데 정적 링크는 #pragma를 사용하거나 개발 IDE를 통해서 연결하므로 라이브러리에서 제공하는 클래스의 접근 및 사용이 용이하다. 하지만 기능 변경 시 라이브러리와 의존적이기 때문에 다시 빌드 및 링크가 필요하여 플러그인 기능 구현에 적합하지 못하다. 동적 링크는 DLL 클래스 사용 시 함수(EXPORT Function) 단위의 접근에 대한 제약이 존재하지만 시스템과 연결이 독립적이므로 규약에 맞는 상황이라면 DLL 교체를 통해서 기능 구성

이 가능하기 때문에 플러그인을 구현할 수 있다. 그림 1, 2는 DLL의 동적 링크를 통해서 클래스의 함수 접근 방법과 제안하는 메시지 프로토콜의 모듈 구조를 보여준다.

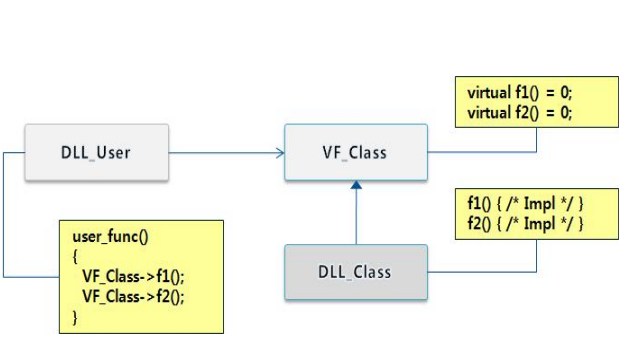

그림 1 DLL 클래스 함수 접근 구조

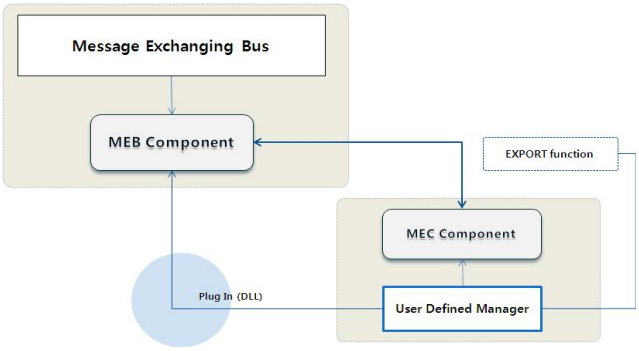

그림 2 제안 메시지 프로토콜 상호작용 구조

프레임워크(DLL_User)에서 접근할 DLL_Class의 함수들을 가상 함수(VF_Class)로 선언함으로써 링크 에러를 방지하며, DLL 클래스의 함수를 일반 함수의 접근 방법과 동일하게 취급할 수 있다. 하지만 DLL과 프레임워크의 의존성을 제거하기 위해서 프레임워크의 객체 생성 없이 서비스에 접근하게 되면 DLL 개발 시 링크 참조 에러가 발생한다. 따라서 프레임워크와 상호작용하는 클래스에 대한 널(Null) 객체를 DLL에 제공함으로써 링크 에러를 제거할 수 있다[그림 3, 4].

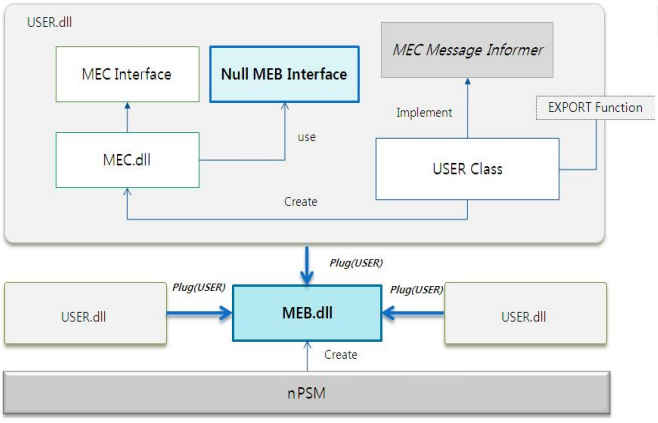

그림 3 NullMEB 모듈을 사용한 플러그인 구조

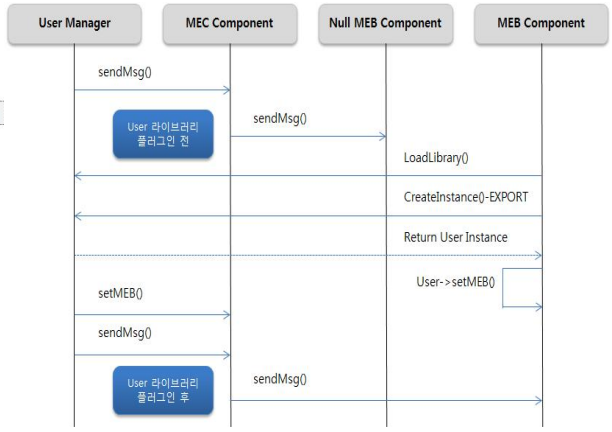

그림 4 플러그인 인터페이스 시퀀스

그림 3과 같이 Null MEB Interface(널 객체)를 사용하면 프레임워크 생성 유무와 관계없이 프레임워크와 상호작용하면서 DLL을 구현할 수 있으며, 프레임워크에 플러그인 되면 생성된 MEB 객체를 바인딩함으로써 서비스를 수행할 수 있다. DLL은 플러그인 시 교환 메시지를 작성한 XML을 프레임워크에 입력하는데 작성된 메시지의 선언 정보(Publish / Subscribe)를 해석하여, 객체 정보가 아닌 데이터 정보를 통해서 라우팅 테이블을 생성한다. 따라서 새로운 DLL 추가를 요구할 경우 데이터를 송신(Publish) 또는 수신(Subscribe) 여부를 교환 데이터에 할당하여 프레임워크 라우팅 테이블에 자동으로 등록하기 때문에 기존 교환 인터페이스에 대한 의존성을 제거할 수 있다.

향후, DLL의 추가 및 삽입 시 목록에 대한 관리 방법이 필요하다. 즉, 파일 목록을 읽어오는 방법이나 GUI를 통해서 직접 읽어오는 방법 등이 연구되어야 한다. 또한, 기존 코드 수준에서 재사용할 때와 DLL로 재사용할 때 프레임워크 구성 요소간의 메시지 교환 성능에 대한 성능이 분석되어야 한다.