http://dx.doi.org/10.3745/KIPSTA.2012.19A.2.101

TMO 기반의 실시간 객체 모델의 코드 자동생성기법 연구

석 미 희[†] · 류 호 동[††] · 이 우 진[†††]

요 약

최근에 분산 실시간 소프트웨어가 다양한 분야에서 중요한 역할을 담당하고 있다. 실시간 소프트웨어는 반응 시간에 대한 시간제약성을 만족하여야 함으로 TMO(Time-triggered, Message-triggered Object), CORBA/RT, RTAI 등과 같은 미들웨어를 이용하여야 한다. 하지만 이러한 실시간 미들웨어에 친숙하지 않은 프로그래머들은 실시간 소프트웨어 개발에 어려움이 있다. 이 연구에서는 이러한 미들웨어에 대한 전문지식 없이도 실시간 소프트웨어 개발이 가능하도록 TMO 기반의 자동 코드 생성 도구를 제안하고자 한다. TMO 특성을 설계 모델에 반영하기 위해, 시간제약사항을 포함하는 SpM과 SvM 메소드를 클래스 다이어그램에 추가하고 상태 머신 다이어그램의 독립 영역으로 분할하여 이들의 행위를 표현한다. TMO 기반 코드 생성기는 이러한 모델 정보를 입력받아서 TMO 클래스 코드를 생성한다. 이러한 자동생성 접근 방법은 TMO에 대한 전문지식이 없더라도 실시간 소프트웨어를 생성할 수 있어, 소프트웨어 개발에 소요되는 비용과 시간을 줄이는 장점이 있다.

키워드 : 실시간 소프트웨어, TMO, 자동 코드 생성

A Study of Automatic Code Generation for TMO-based Real-time Object Model

Mi Heui Seok[†] · Ho Dong Ryu[††] · Woo Jin Lee[†††]

ABSTRACT

In recently years, distributed real-time software has performed important roles in various areas. Real-time applications should be performed with satisfying strict constraints on response time. Usually real-time applications are developed on the real-time supporting middleware such as TMO(Time-triggered, Message-triggered Object), CORBA/RT, and RTAI. However, it is not easy to develop applications using them since these real-time middleware are unfamiliar to programmers.

In this paper, we propose an automatic code generator for real-time application based on TMO in order to reduce development costs. For increasing or reflecting the characteristics of TMO into the design model, SpM and SvM methods are added into the class diagram, which have time constraints as their properties. And behaviors of them are represented as separated regions on state machine diagram in different abstract level. These diagrams are inputted into TMO-based code automatic generator, which generates details of the TMO class. Our approach has advantages for decreasing effort and time for making real time software by automatically generating TMO codes without detailed knowledge of TMO.

Keywords : Real-Time Software, TMO, Automatic Code Generator

1. 서 론

오늘날 로봇, 공정 처리, 엘리베이터와 같은 산업용 전기 전자 분야에서부터 통신, 교통관제, 무기체계, 우주항공, 가전제품 분야 등 다양한 분야에 컴퓨터를 이용한 제어 시스템의 도입이 증가함에 따라 분산 실시간 소프트웨어에 대한 요구가 증가하는 추세에 있다[1]. 실시간 소프트웨어는 계산의 논리적 결과가 정확해야 할 뿐만 아니라 처리되는 결과들이 주어진 시간 내에 도출되어야 하는 시간제약조건을 만족하여야 한다. 또한 내장된 시스템의 운용환경은 매우 복잡해지고 있으며 그러한 시스템에 대한 의존도도 가속적으로 심화되고 있다. 이에 따라, CORBA/RT[2], TMO(Time-triggered Message-triggered Object)[3], RTAI(Real Time Application Interface for Linux) [4] 등의 실시간 미들웨어를 활용하여 어플리케이션을 개발하고 있다.

실시간 소프트웨어를 개발하기 위해서 이러한 실시간 미

※ 본 연구는 지식경제부 및 정보통신산업진흥원의 IT융합 고급인력과정 지원사업의 연구결과로 수행되었음(NIPA-2012-C6150-1202-0011).
† 정 회 원 : LG전자 HA사업부 제어연구소 연구원
†† 준 회 원 : 경북대학교 전자전기컴퓨터학부 박사과정
††† 정 회 원 : 경북대학교 IT대학 컴퓨터학부 부교수
논문접수 : 2011년 11월 9일
수 정 일 : 1차 2011년 12월 30일
심사완료 : 2012년 1월 8일

들웨어에 대해 익혀야 하고 소프트웨어가 내장된 시스템을 하위 단계에서 제어를 하여야 함으로 실시간 소프트웨어 프로그래밍에 어려움이 따른다. 복잡한 플랫폼이나 미들웨어를 활용하여야 하는 분야에서는 모델 기반 코드 생성 기법을 많이 활용한다. 자동차 분야의 AUTOSAR 가 대표적인 예이다. 이 연구에서는 TMO 모델 기반의 코드 생성 기법을 제안한다. TMO는 정시 보장, 객체 지향, 분산 환경 등의 특징을 통합하는 대표적인 분산 실시간 객체 모델로 경성 또는 연성 실시간 응용에서 사용될 수 있어 TMO 미들웨어를 선택하였다. TMO 미들웨어로는 윈도우를 지원하는 UCI의 WTMOS(Windows TMO System)와 리눅스 환경을 지원하는 LTMOS(Linux TMO System), 커널 형태로는 리눅스 커널을 수정하여 커널 API와 내장 실시간 스케줄러로 직접 분산 실시간 컴퓨팅을 지원하는 TMO-Linux와 임베디드 커널용인 TMO-eCos가 있다[5]. 본 논문에서는 WTMOS를 이용해 윈도우 상에서 실시간 시스템을 지원하는 TMO 객체기반 코드 자동 생성기의 설계를 제안한다.

이 연구에서는 실시간 UML(Unified Modeling Language)[6]을 일부 응용해 구조적인 측면에서 클래스 다이어그램을 입력 받고 행위적인 측면에서 상태 머신 다이어그램을 입력으로 받아 완성된 TMO 객체 기반의 실시간 소프트웨어 코드를 생성하는 것을 제안한다. (그림 1)과 같이, 먼저 TMO 기반 모델링을 지원하기 위해서 클래스 다이어그램과 상태 머신 다이어그램을 확장한다. 그리고 TMO 기반 모델을 바탕으로 TMO 코드를 자동 생성하는 방법을 제안한다. 일반 클래스와 달리 TMO 클래스는 시간 제약 조건이 존재하는 SvM 메소드와 SpM 메소드를 갖고 있다. 이러한 정보들을 클래스 다이어그램에 추가하고 시간 제약 조건을 위한 추가적인 노드를 연결해 보이고 구현상에서 각 메소드의 속성으로 추가한다. 또한 상태 머신 다이어그램에서의 TMO 모델링은 기존의 표기법을 그대로 응용하여 영역을 나누는 방식으로 표현한다. 기존의 상태 머신 다이어그램은 하나의 영역에서 하나의 객체의 흐름을 표현하지만, TMO 객체는 각자 다른 방식으로 존재하는 두 개의 메소드를 각자 다른 영역에 표현한다.

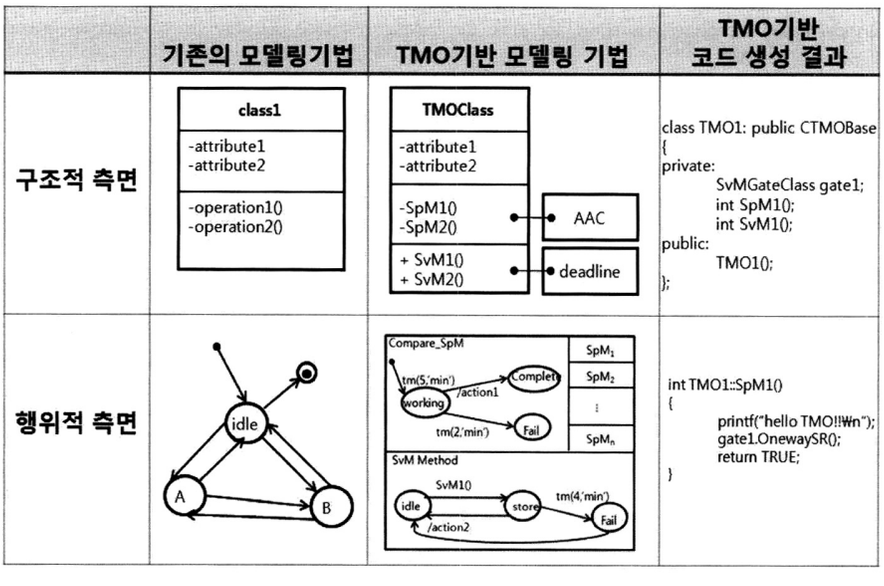
이 논문의 구성은 다음과 같다. 제 2 장에서 실시간 소프트웨어의 특성과 TMO 객체, 지금까지 나온 다른 자동 코드 생성기의 특징에 대하여 설명한다. 제 3 장에서는 이 연구에서 제시하는 TMO의 특징을 반영한 클래스 다이어그램과 상태 머신 다이어그램의 모델링 기법에 대하여 설명하고 다이어그램 편집기의 생성 방법을 설명한다. 제 4 장에서는 소스코드 자동 생성기의 세부적인 실행 과정과 구조에 대하여 설명한다. 제 5 장에서는 구현된 화면 및 시범 적용 과정을 보이고 마지막으로 제 6 장에서는 결론을 기술한다.

2. 연구 배경

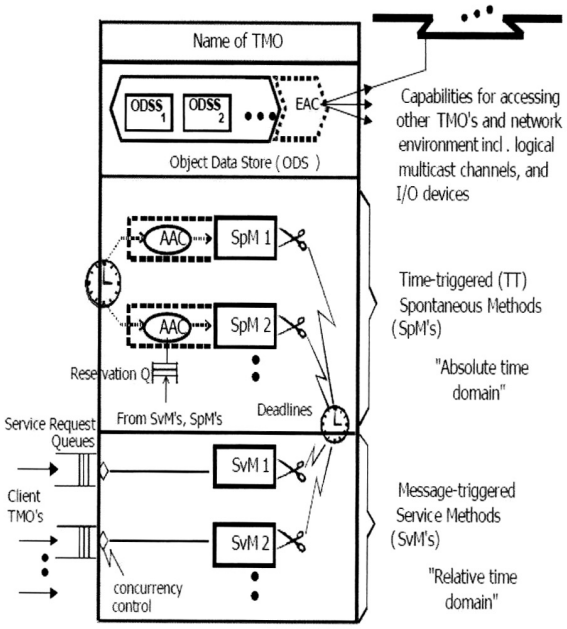
이 장에서는 윈도우상에서 사용하는 실시간 TMO 객체인 WTMOS에 대해서 설명하면서 TMO 객체의 어떠한 특징이 모델링과 기존 도구에서 코드 자동 생성하는 것을 어렵게 하는지 알아보고 코드 자동 생성 기능을 가진 Rational Rhapsody 도구에 대해서 살펴본다.

2.1 TMO 객체

TMO(Time-Triggered Message-Triggered Object) 객체[3]는 UCI의 Dream Lab에서 개발한 객체 모델로서 객체 모델을 실시간 시스템에 적합하도록 확장한 것이다. TMO 객체는 기존 객체 모델을 확장한 것으로 객체 자체적으로 실시간 특성을 보장하고 C++ 라이브러리로 구현된 실시간 객체이며 분산 환경을 지원한다. 기존의 객체는 클라이언트의 서비스 요청 메시지에 의해 동작하지만 TMO는 객체 특성에 자치적인 동작을 추가한 것이다. TMO의 특성은 분산 컴퓨팅 객체로써 원격 메소드의 호출을 통해서 상호동작 가능하다.



(그림 1) TMO 기반의 모델링 및 코드생성의 개요



(그림 2) TMO 객체의 구조

TMO 객체의 기본 구조는 (그림 2)와 같고 내부구조는 게 ODS(Object Data Store), SpM(Spontaneous Method), vM(Service Method)의 세 부분으로 구성된다. (그림 2)의 부분을 살펴보면 ODS 영역은 객체의 정보를 저장하는 역을 의미하는 것으로 객체들의 데이터 멤버들로 이루어 있고 EAC(Environment Access Capability)에서 원격 객 호출과 입출력 장치 인터페이스를 지원하는 통신채널이 공된다. SpM 메소드는 시간 구동 메소드로 주어진 시간 건에 의해 자율적, 주기적으로 구동된다. SpM 메소드에는 간 조건인 AAC(Autonomous Activation Condition)가 주 지고 이에 따라 구동하게 된다. AAC는 시작과 종료시간, 기, 데드라인 등으로 구성된다. AAC는 SpM 메소드의 시 제약 조건을 표현하기 위한 구조체로 선언된다. SvM 메 드는 외부 또는 내부 클라이언트의 메시지 요청에 의해 동하게 되는 메소드이다. SvM 메소드는 서비스 완료 시 지의 데드라인이 적용된다. SvM 메소드는 항상 게이트를 해 호출된다. TMOSM(TMO Support Middleware)은 분 실시간 객체인 TMO를 지원하기 위한 미들웨어로 실시 서비스를 플랫폼이나 운영체제의 제약 없이 지원한다. 렇게 시간 제약 조건이 구조체로 구현되고 각 클래스와 메소드를 미들웨어에 등록해야 하는 것 등이 다른 언어들과 른 점을 보여 기존의 코드 자동 생성기에서 코드를 생성 는 것을 어렵게 한다.

TMO를 사용하기 위해서는 TMOSL.h 파일을 포함해서 용하고 이외의 라이브러리 파일을 참조해서 사용하여야 한다. 또한 config.ini 파일에서 분산 환경을 설정하는 것을 는다. 이러한 파일들은 UCI의 Dream Lab 홈페이지에서 제공되며 config.ini 파일을 생성하는 프로그램 또한 제공 린다.

2.2 코드 생성 관련 연구

UML 모델을 이용하여 소스 코드를 생성하는 연구는 많이 있어 왔다. UML 모델에서 C++ 코드와 Ada95 코드를 생성하는 연구[7], Java 코드를 생성하는 연구[8], VHDL 코드를 생성하는 연구[9], 그리고 Simulink와 유한상태머신과 같은 다른 모델로 변환하는 연구[10] 등의 다양한 연구들이 수행되어 왔으며, 또한 이러한 기법들을 지원하는 Rhapsody[11], JComposer[12] 등의 다양한 상용도구들도 존재한다.

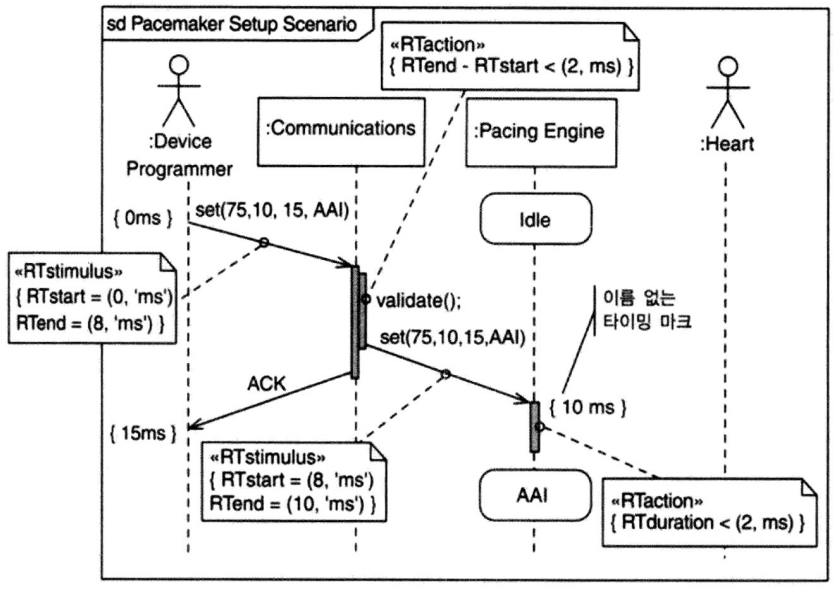
여러 소스 코드 자동 생성기가 있지만 가장 대표적인 상용 도구로는 IBM의 Rational Rhapsody가 있다. Rhapsody는 C, C++, Java, Ada 등의 언어를 지원하며 UML의 그래픽 프로그래밍과 통합되어 실제 코드 생성을 통한 테스트까지 제공하여 디자인은 곧 해석이라는 완벽한 개발 환경을 제공한다. 또한 최고의 기술을 사용하여 팀 간의 협력과 재사용 가능한 디자인의 개발, 소프트웨어의 질을 향상시키는데 많은 도움을 준다. Rhapsody는 개발 프로세스의 획기적인 발전과 개발 기간의 단축으로 전체 개발비용을 줄일 수 있으며 궁극적으로 시장 적응력을 향상시켜줄 하나의 도구로 자리매김하였다. 디자인 패턴을 C, C++언어로의 생성이 가능하다는 장점이 있지만, 실시간 시스템에 적용하기에는 적절치 못하였다. 실시간 시스템을 위한 코드를 생성하려면 상태머신 다이어그램에서 시간관련 코드를 모두 입력해야 하는 번거로움이 존재한다. Rhapsody 외에도 시각화 도구를 이용해 모델을 입력받아 소스코드를 자동으로 생성하게 하는 도구인 JComposer가 존재한다. JComposer는 Oris도구[13]에서 제공하는 시간정보가 내장된 시각적인 모델 또는 선점 Petri-net을 입력받아 xml 파일로 변환하고 이 변환된 xml 파일을 JComposer에서 제공하는 XSLT(Extensible Stylesheet Language Transformations)를 이용해 RTAI 코드를 자동으로 생성하는 것을 지원한다.

앞서 제시한 도구들은 모두 모델 주도 개발을 통한 코드 생성을 이용하여 소프트웨어의 신뢰성과 품질을 높이려고 노력해왔다. 이를 위해 모델링 과정에서 지원되는 여러 부가 기능과 생성되는 코드의 최적화에 대한 많은 노력을 기울여왔다. 하지만 본 논문에서는 기존의 도구에서는 실시간성을 스스로 보장하는 TMO 객체를 표준 UML을 사용해 모델링하고 TMO 객체기반 코드의 구현이 가능한 코드 생성기를 제안한다.

3. 실시간 TMO 객체의 특징을 반영한 모델링 기법

3.1 실시간 특성을 반영한 모델링

기존 UML(Unified Modeling Language)[14]은 어떠한 플랫폼에서도 모델링이 가능하다는 장점이 있지만 시간제약 조건에 대한 기술방법은 지원하지 않는다. 이를 지원하기 위해 OMG에서는 실시간 UML[6]을 제안하였다. 이 연구에서는 실시간 UML을 바탕으로 TMO 객체의 실시간 특성을 모델링한다. 실시간 시스템인 TMO 객체를 지원하기 위해 OMG에서 제공하는 시간표기법(타이밍 마크)을 응용한다[15].



(그림 3) 실시간 UML의 타이밍 마크의 예

(그림 3)은 타이밍 마크를 사용해 동작 실행에 타이밍 정보를 적용한 실시간 UML의 예제이다[6]. 그림에서 <<RTaction>>은 스테레오 타입으로 시간이 소요되는 동작을 의미하며, <<RTstimulus>>는 시간이 정해진 동작을 나타낸다. <<RTstimulus>>의 RTstart는 동작이 시작하거나 자극을 보낸 시간, RTend는 동작이 끝나거나 자극에 대한 응답이 완료된 시간을 의미한다. <<RTaction>> 의 RTduration은 동작의 실행이나 자극 응답 시간의 길이를 나타낸다. 즉, RTduration = RTend - RTstart를 의미한다. 괄호 안의 숫자와 문자는 시간을 의미하여 시간 단위 <ms, s, min, hr, days, wks, mos, yrs>등으로 표현된다.

3.2 클래스 다이어그램의 TMO 확장

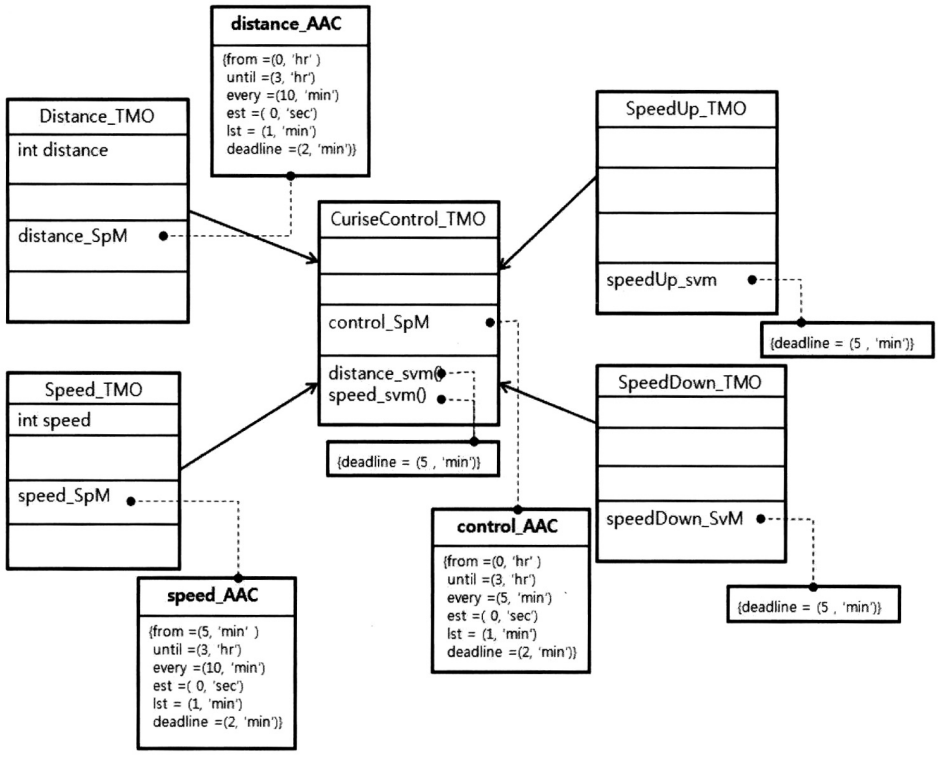
클래스 다이어그램은 시스템의 구조적인 설계를 그래픽으로 표현한 것이다. 클래스 다이어그램의 구성요소로는 클래스와 인터페이스, 그리고 관계를 나타내는 의존 관계, 상속 관계, 연관 관계 등이 있다. 기존의 클래스는 이름과 속성, 그리고 메소드(Operation)으로 구성되어 있다. 그러나 이 연구에서는 기존의 클래스 다이어그램과 달리 시간 제약조건에 의해 동작하는 메소드들인 SpM 메소드와 SvM 메소드를 사용하기에 이름과 속성, 일반 메소드 외에도 SpM 메소드와 SvM 메소드를 클래스의 구성요소로 추가한다. 따라서 클래스 다이어그램의 속성으로 속성과 일반 메소드, SpM, SvM이 나타난다.

TMO 클래스는 TMO 라이브러리에서 제공하는 CTMOBase 클래스를 상속받아 사용하기에 일반 클래스와 따로 정의하여 추가하고 기존의 메소드와 달리 SpM 메소드와 SvM 메소드는 시간 제약 조건이 추가되므로 각 메소드의 속성으로 시간 제약 조건이 추가로 구성된다. 이를 바탕으로 코드 자동 생성기에서는 TMO 클래스가 모델도구에 입력될 경우 자동으로 CTMOBase 클래스를 상속받도록 설계하고 SpM 메소드와 시간 제약 조건을 받아 자동으로 AAC 구조체를 생성할 것이다.

(그림 4)는 실시간 UML의 표기법을 응용해 TMO 객체 기반으로 확장한 순항 제어(Cruise Control) 시스템을 클래스 다이어그램으로 나타낸 것이다. 순항 제어 시스템은 먼저 Distance_TMO와 Speed_TMO 시스템에서 앞차와의 거리와 현재 속도를 측정한 후, Cruise_Controller_TMO 시스템에 전송한다. 이후 Cruise_Controller_TMO 시스템은 측정된 거리와 속도를 바탕으로 가속해야 할 경우에는 SpeedUp_TMO 시스템을 호출해 가속하고 감속해야 할 경우에는 SpeedDown_TMO 시스템을 호출해 감속하는 시스템이다.

(그림 4)의 클래스 다이어그램에서 Cruise_Controller_TMO 클래스를 보면 control_svm의 시간 제약 조건을 표현한 control_AAC 노드가 추가적으로 존재하며 distance_svm과 speed_svm 메소드의 시간 제약 조건을 표현한 deadline 노드가 추가로 연결되어있다. 세부적으로 AAC 영역에서는 from, until, every, est, lst, deadline의 시간정보가 포함된 것을 볼 수 있다. from은 SpM 메소드의 시작시간을 나타내고 until은 동작 기간, every는 주기, est(early start time)과 lst(latest start time)로 명시된 시간사이에는 실행되어야 한다는 것을 의미한다. deadline은 서비스 수행이 이 시간 안에는 끝나야 한다는 것을 명시한다. SvM 메소드의 deadline도 SpM 메소드와 같은 의미로 수행이 완료되어야 하는 시간을 의미한다.

Cruise_Controller_TMO 클래스의 control_SpM 메소드는 "시스템은 시작 후 3시간 동안 매 5분마다 다시 동작하고 시작한 시점부터 1분이 지나기 전까지는 수행되어야 하며 시작한 시점부터 2분 안에 동작을 마쳐야 한다."는 것을 의



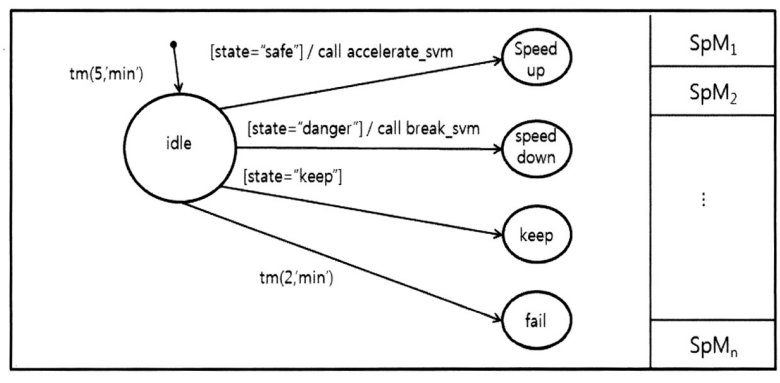
(그림 4) 순항 제어 시스템의 TMO 확장 클래스 다이어그램

미한다. distance_SvM 메소드는 "호출된 시점부터 5분 안에 동작을 마쳐야 한다."는 것을 의미한다. 이처럼 RTUML에서는 메모의 형태로 시간 제약사항을 나타냈지만 TMO 클래스의 경우 필수적인 요소를 RTUML의 표기법을 응용해 위와 같이 노드로 나타내어 확장하고 이를 바탕으로 코드 자동 생성기를 구현한다.

3.3 상태 머신 다이어그램의 TMO 확장

이 절에서는 상태 머신 다이어그램을 이용하여 시스템의 동적 측면을 모델링한다. 상태 머신 다이어그램은 클래스나 전체 시스템의 생명주기를 명시하는 일과 관련이 있다. 상태 머신에서 나타내는 전이는 어떤 개체가 첫 번째 상태에서 특정 동작을 수행한 후, 두 번째 상태로 들어가는 두 상태간의 관계로서 다만 명시된 상태가 발생하고 명시된 조건이 만족될 때만 진행한다. 하지만 TMO 객체는 특정 시간이 되면 자동으로 상태가 전이하게 되는 SpM 메소드와 기존의 개체와 같이 외부의 호출이 발생할 경우에 상태가 전이하게 되는 SvM 메소드가 존재한다. 또한 두 메소드는 일정 시간이 지나면 자동적으로 메소드의 상태가 전이되므로 이를 표현할 수 있어야 한다.

이 연구에서는 기존의 상태머신 다이어그램의 표기법을 그대로 사용하고 SpM 메소드와 SvM 메소드를 각 메소드의 영역에서 나타내고 각 영역은 표현의 추상화 수준을 다르게 표현한다. SvM은 외부에 노출되는 메소드이므로 SvM 메소드가 호출되는 사건이 발생하면 상태가 전이한다. 따라서 SvM 메소드 영역에서는 클래스의 상태 변화가 나타나도



(그림 5) Control_spm 메소드의 상태 머신 다이어그램

록 모델링한다. 그러나 SvM 메소드와 달리 SpM 메소드는 외부에서 호출되는 메소드가 아니기 때문에 메소드 내부의 흐름을 나타내도록 모델링한다. 이렇게 서로 다른 추상화 수준을 가지기에 SpM 메소드와 SvM 메소드는 하나의 상태머신에서 영역을 나누어 병렬적으로 나타낸다.

(그림 5)는 CruiseControl_TMO 클래스를 상태머신 다이어그램으로 표현한 예를 보인다. 먼저 SpM 메소드 영역을 상태머신 다이어그램으로 나타난 예를 보인다. control_spm 메소드 영역을 보면 control_spm 메소드는 5분마다 호출되어 수행하고 데드라인인 2분의 시간을 넘기게 되면 fail 상태가 된다. 주기적으로 상태를 체크하여 상태가 safe일 경우 속도를 유지하기 위해 액션을 취하지 않는다.

또한 상태가 speed up일 경우 Accelerate_TMO 클래스의 accelerate_svm을 호출해 가속하고 상태가 danger이거나 speed down일 경우 Accelerate_TMO 클래스의 break_svm을 호출해 감속한다. 만약 SpM 메소드가 여러개 존재할 경우 영역을 추가해 각 영역별로 메소드의 내부흐름을 나타낸다.

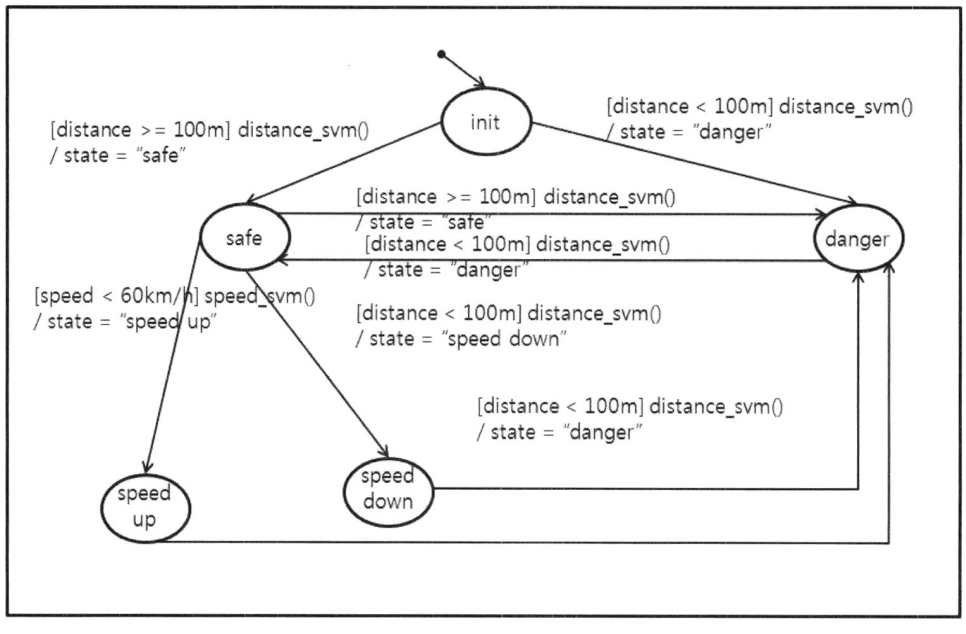
반면 SvM 메소드의 경우 SpM 메소드와 달리 클래스의 상태 변화가 나타나기에 하나의 영역에 모든 SvM 메소드가 나타난다. (그림 6)은 SvM 메소드 영역을 상태머신 다이어그램으로 나타난 예를 그림으로 보여준다. 그림에서 svm 영역을 보면 "[조건] 메시지(메소드의 이름) / 액션"이 트랜지션 상에 나타나는 것을 볼 수 있다. SpM 메소드의 경우 하나의 영역이 메소드를 나타내기에 메소드의 이름이 메시지로 나타날 필요가 없지만, SvM 메소드의 경우 하나의 영역에 모든 SvM 메소드가 나타나기에 메시지에서 메소드의 이름이 필요하다.

(그림 6)의 상태머신 다이어그램을 살펴보면, distance_svm 메소드가 호출될 경우 상태가 init이면 Distance_TMO에서 전송한 거리 데이터를 비교해 100미터보다 가까우면 상태를 danger로 바꾸는 액션을 취하고 danger 상태로 전이된다. 만약 거리가 100미터가 넘을 경우 상태를 safe로 바꾸는 액션을 취하고 safe 상태로 전이한다. speed_svm 메소드의 경우에 Speed_TMO의 speed_spm 메소드가 호출 될 경우에 시작하고 속도에 따라 speed up 상태와 speed down 상태로 전이하게 된다.

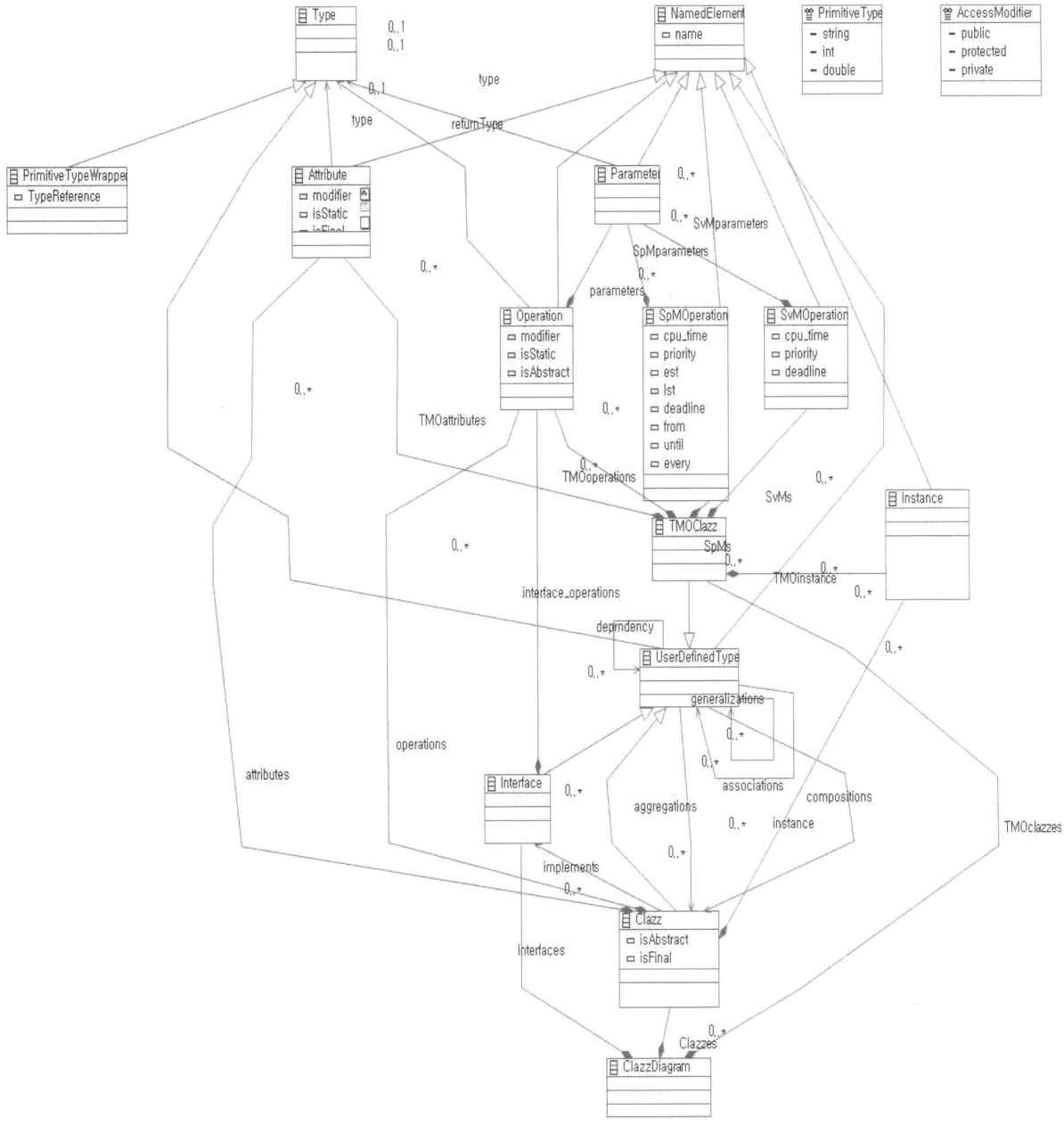
3.4 TMO기반 모델링 도구

오픈 소스 기반의 이클립스(Eclipse)[16]는 응용 도구 개발을 지원하기 위해 다양한 플러그인들을 제공하고 있다. 플러그인 중 다이어그램 편집기를 개발하기 위해 사용되는 주요한 플러그 인으로는 Workbench, FJace, SWT(Standard Widget Toolkit), EMF(Eclipse Modeling Framework), GMF(Graphical Modeling Framework) [17] 등이 있다. 이 연구에서는 GMF 프레임워크 기반 다이어그램 편집기 개발 기법[18]을 이용하여 TMO 기반 클래스 다이어그램과 상태머신 다이어그램의 편집기를 개발한다. (그림 7)은 기존의 클래스 다이어그램에 TMO 클래스를 추가하여 메타모델을 정의한 것으로, (그림 7)의 메타모델을 사용하여 다이어그램 편집기를 개발할 경우 SpM 메소드와 SvM 메소드, 그리고 각 메소드의 시간정보를 편집기 창에서 입력할 수 있다. GMF를 사용하여 그래픽 부분 모양, 노드, 링크 등을 정의한다. 기존의 클래스 다이어그램과 달리 TMO 클래스에는 SpM 메소드와 SvM 메소드의 구획이 추가된다.

상태 머신 다이어그램 편집기는 이클립스 GMF 플러그인에서 제공하는 도구를 상속받아 사용한다. 클래스 다이어그램 편집기에서 클래스를 생성한 후, 클래스의 자식 노드로 상태 머신 다이어그램을 생성할 수 있다. 이렇게 하면 클래스 하나에 상태 머신 다이어그램 하나가 생성된다.



(그림 6) Control_SpM 메소드의 상태 머신 다이어그램



(그림 7) TMO 지원 확장된 클래스 다이어그램의 메타 모델

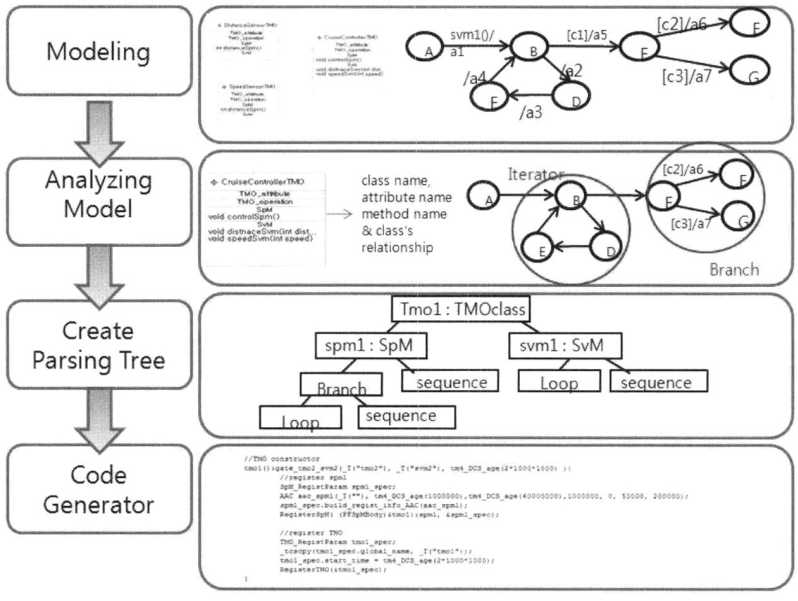
4. TMO 기반 코드 자동 생성기의 설계

이 연구에서 제시하는 코드 자동 생성기는 크게 다이어그램 편집기와 코드 자동 생성기의 두 부분으로 나누어져 있다. 다이어그램 편집기는 앞에서 설명한 바와 같이 클래스 다이어그램 편집기와 상태 머신 다이어그램 편집기로 나누어진다. 이렇게 두 가지 다이어그램의 입력을 통하여 코드를 생성한다.

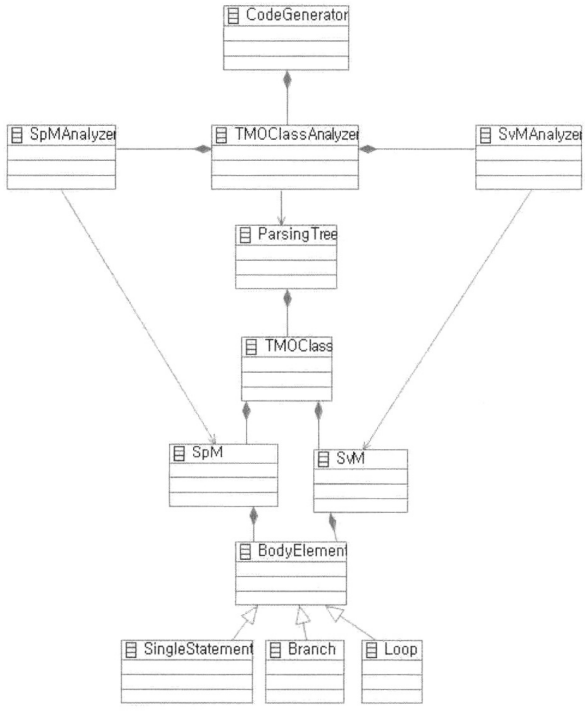
4.1 TMO 객체기반 코드 자동 생성

코드 자동 생성과정은 (그림 8)과 같이 4단계를 거친다. 먼저 모델링 과정을 통해 모델이 생성되면, 모델을 입력받아 모델 분석기를 통해 모델을 분석한다. 클래스 다이어그램 분석기로 클래스와 속성, 메소드를 분석하고 상태 머신 다이어그램 분석기로 SpM 메소드와 SvM 메소드의 내용을 분석한다. 이후 분석된 정보로 모델 파싱 트리를 생성하고 파싱 트리를 순회하면서 코드가 생성된다.

이를 위해서는 (그림 9)와 같은 구조가 필요하다. (그림 9)는 이 연구에서 제시하는 코드 자동 생성기의 설계도로서 전체 구조를 나타낸 것이다. 위의 그림과 같은 구조로 코드를 생성하면, 먼저 클래스 다이어그램에서 입력 받은 정보로 클래스의 골격이 되는 코드를 생성하게 되고 클래스 및 함수의 행위는 상태 머신 다이어그램을 분석하여 코드를 생성된다. 본 논문에서 제안하는 코드 자동 생성기는 위의 구조에 따라 구현한다.



(그림 8) TMO 객체 모델 기반 코드생성 과정



(그림 9) TMO 기반 코드 자동 생성기의 클래스 다이어그램

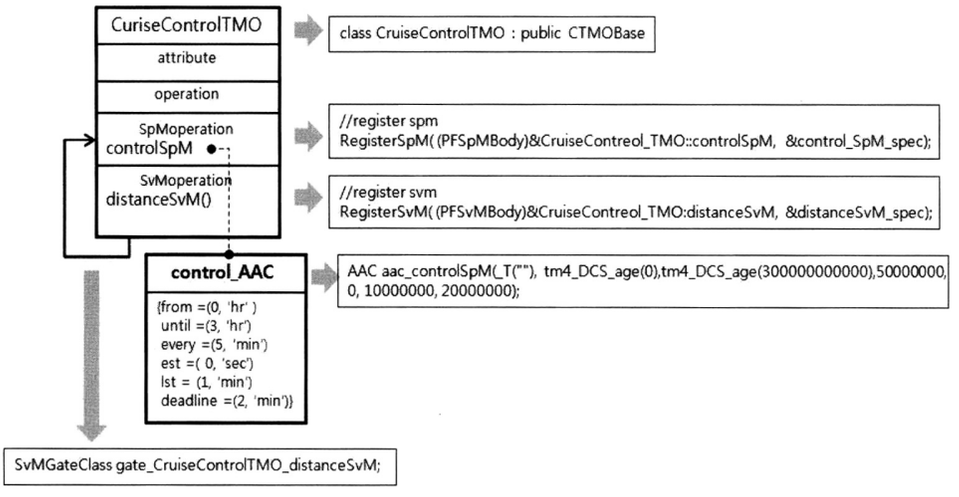
4.2 클래스 다이어그램에서의 코드 생성 규칙

이 절에서는 클래스 다이어그램에서의 코드 생성 규칙에 대해 설명한다. 클래스 다이어그램에서 얻어올 수 있는 정보는 크게 클래스 이름과 속성 그리고 메소드 이름, 클래스 간의 관계이다. 세부적으로 살펴보면 TMO 클래스의 경우 일반 클래스와 달리 Dream Lab에서 제공하는 TMO 라이브러리를 상속받아 사용하게 된다. 따라서 TMO 클래스가 다이어그램 모델에 입력으로 들어올 경우 실제 코드에 자동적으로 TMOSL.h 파일을 추가하고 TMO 클래스는 CTMOBase 클래스를 자동적으로 상속받도록 구현한다. 또한 TMO 객체는 TMO 클래스의 생성자에서 메소드와 객체를 미들웨어에 등록하는 과정을 거치게 되는데 이러한 부분은 자동적으로 추가되도록 한다.

SpM 메소드 또한 TMO 클래스와 마찬가지로 미들웨어에 등록하는 코드가 필요하다. SpM 메소드의 경우 클래스 다이어그램 편집기에서 시간 제약 정보를 입력받는다. 이렇게 입력받은 정보는 메소드를 미들웨어에 등록할 때 시간 제약 정보(AAC)를 함께 등록한다. 따라서 SpM 메소드가 존재할 경우 자동적으로 미들웨어에 등록하는 코드를 추가하고 사용자가 지정한 AAC 정보를 등록한다. SvM 메소드도 SpM 메소드와 마찬가지로 미들웨어에 등록하는 과정이 필요하기에 미들웨어에 등록하는 코드를 추가하고 사용자가 클래스 다이어그램에 입력한 시간정보를 등록하는 코드를 생성한다.

SvM 메소드의 경우 호출하기 위해서는 SvM 메소드별로 할당된 게이트가 반드시 필요하다. 만약 tmo1 클래스에서 tmo1 클래스가 가지고 있는 SvM1 메소드를 호출하기 위해서는 tmo1 클래스는 SvM1 메소드를 호출할 게이트가 필요하다. tmo1에서 tmo2가 가지고 있는 SvM2 메소드를 호출하기 위해서는 tmo1에서는 SvM2를 호출할 게이트가 필요하다. 이렇게 게이트를 사용하기 위해서는 TMO 클래스의 속성으로 게이트 클래스를 생성해야 하고 클래스 사이에 연관관계가 있으면 호출할 대상 SvM 메소드를 호출하는 게이트를 추가한다. 이때의 게이트 클래스의 이름은 "gate_호출할 TMO 클래스이름_호출할 SvM 메소드이름"으로 생성되도록 구현한다. 이러한 규칙을 바탕으로 모델 구조 분석기에서 클래스 다이어그램을 분석한다.

(그림 10)은 TMO 클래스는 CTMOBase 클래스를 상속



(그림 10) 클래스 다이어그램에서의 코드 생성 규칙

받고 SpM 메소드와 SvM 메소드는 미들웨어에 등록하는 것을 보여준다. 이것들은 TMO 클래스의 생성자에서 처리한다. 또한 연관관계가 있을 경우 자동으로 게이트를 생성하고 시간 제약 조건의 정보를 입력받아 AAC 정보를 자동으로 생성하는 것을 볼 수 있다.

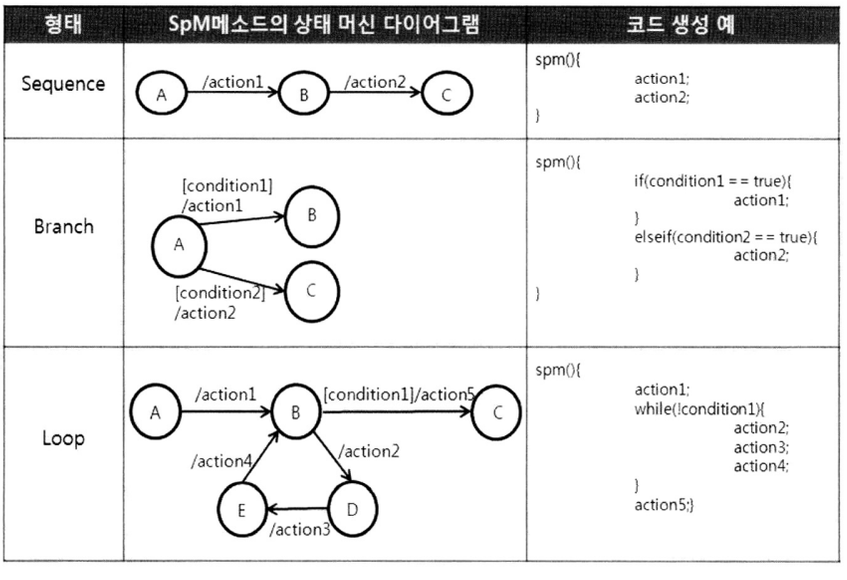
4.3 상태 머신 다이어그램에서의 코드 생성 규칙

TMO 클래스에는 일반 메소드와 다른 특성을 가진 SpM 메소드와 SvM 메소드가 존재하므로 이러한 메소드의 코드를 생성 방식에도 차이가 발생한다. 이 절에서는 상태 머신 다이어그램에서의 코드 생성 규칙을 설명한다.

4.3.1 SpM 메소드 코드 생성 규칙

SpM 메소드는 상태 머신 다이어그램의 한 영역이 하나의 메소드를 나타낸다. 따라서 SpM 메소드의 이름과 영역의 이름은 일치해야 한다. 이렇게 메소드와 영역을 일치하게 되면, 상태머신 다이어그램의 트랜지션 상에 메시지(메소드의 이름)이 나오지 않게 된다. 이 부분은 SvM 메소드와 다른 부분이므로 유의하여야 한다. SpM 메소드 구조 분석기에서는 입력된 모델에서의 패턴을 분석한다. (그림 11)은 몇 가지 기본 패턴과, 각 패턴들이 실제로 코드 상에서 어떻게 구현되는지를 보인다.

(그림 11)에서는 가장 일반적인 패턴인 순차 패턴, 조건문을 통한 분기, 단순하게 반복되는 패턴에 대하여 설명하고 있다. 이 연구에서 제안하는 코드 자동 생성기는 트랜지션 상에 코드가 들어가므로 상태와 트랜지션 상에 입력된 조건, 그리고 액션으로 나타나는 코드를 참고하게 된다. 일반적으로 분기문은 분기를 만나면 if-else 문을 삽입하게 되고 '/' 뒤의 액션에 코드가 들어간다. 반복문 같은 경우에는 while 문으로 선언된다. 반복문은 각 상태를 트랜지션을 따라가며



(그림 11) 상태 머신 다이어그램에서의 SpM 메소드의 코드 생성 규칙

방문하고 방문한 상태가 여러번 반복이 될 경우 반복문으로 정의한다. 위의 모델링과 달리 코드 생성을 위한 상태머신 다이어그램상에서는 데드라인을 위반한 fail 상태를 추가하지 않는다. 시간이 지나면 자동적으로 상태가 전이되므로 추가로 정의하지 않는다.

4.3.2 SvM 메소드 코드 생성 규칙

SvM 메소드는 클래스 전체의 흐름을 갖게 되고 SpM 메소드와 달리 하나의 영역에 모든 메소드의 행위가 나타난다. 따라서 영역의 이름과 메소드의 이름의 일치성을 갖고 메소드의 내용을 생성하던 SpM 메소드와 달리 트랜지션 상의 메시지를 참고해 각 메소드의 내용을 생성하게 된다. 트랜지션 상에서 "[조건]메시지/액션"의 순서로 조건과 메소드 이름 그리고 수행할 코드가 입력된다.

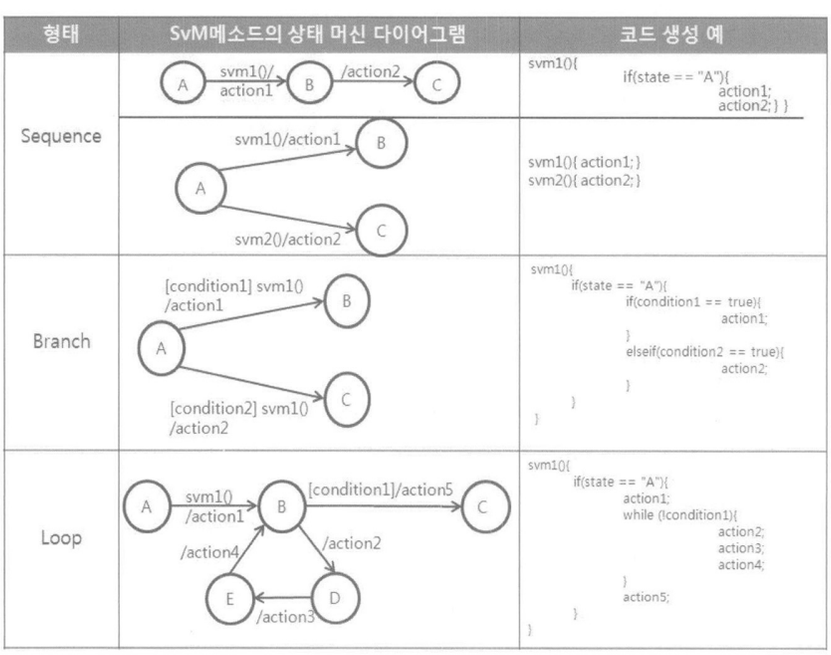
SvM 메소드 구조 분석기에서는 입력된 모델에서의 패턴을 분석한다. (그림 12)는 SvM 메소드의 몇 가지 기본 패턴과, 각 패턴들이 실제로 코드 상에서 어떻게 구현되는 지를 보인다.

먼저 Sequence의 상태 머신 다이어그램의 예를 보면, 하나의 상태에서 두 개의 트랜지션이 나타나는 것을 볼 수 있다. 이 두 개의 트랜지션은 각자 다른 메소드의 내용을 나타내는 것을 트랜지션 상에서 나타나는 메시지를 보고 판단한다. 따라서 위의 SpM 메소드처럼 트랜지션이 여러 개 나타나도 if-else 문을 생성하지 않는다. 두 번째로 분기문을 보면 조건과 메소드 이름, 그리고 액션을 모두 반영하여 if-else 문을 생성한다. 반복문이 나타날 경우 트랜지션 상의 반복 부분에서는 한 번의 메소드 호출만이 있어야 한다. SvM 메소드의 반복문 또한 트랜지션을 따라가며 방문한 상태를 저장하고 저장된 상태가 반복되면 while 문을 생성한다.

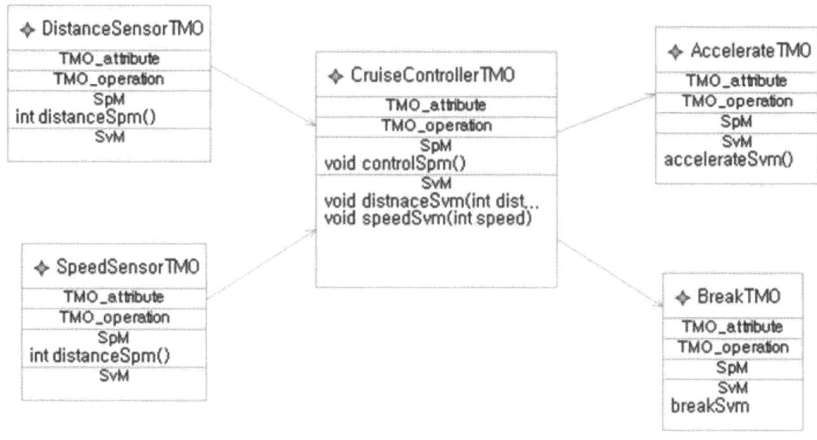
if(state == "A")는 상태가 변화하는 것을 코드로 나타낸 것으로, 본 논문의 자동 생성기에서는 현재 상태를 저장하는 클래스 속성 상태를 자동으로 생성한다. 클래스의 메소드가 호출되면 이 메소드에서는 클래스의 현재 상태에 따라 그 행위가 결정된다. 이를 위해서 코드 생성기는 메소드를 생성할 때 if(state == "현재상태") { action; state = "다음상태";}와 같은 코드를 생성한다. 하지만 트랜지션에 액션만이 존재할 경우 상태에 머무르지 않기에 다음 상태를 저장하지 않는다.

5. 구현 및 시범 적용

이 연구에서 개발한 주요 모듈은 TMO 특성을 확장한 클래스 다이어그램 편집기와 클래스 정보를 입력으로 받아 코드를 자동 생성하는 코드자동생성기이다. 확장된 클래스 다이어그램 편집기는 이클립스 GMF 환경을 이용하여 생성하였다. 이클립스 GMF 환경에서는 다이어그램 편집기의 메타모델을 정의하고 GMF에서의 다이어그램 생성과정[14]을 따라 편집기 플러그인을 생성한다. 코드자동생성기에서는 클래스 다이어그램 정보와 상태머신 다이어그램 정보가 들어있는 이클립스 EMF 모델에서 정보를 추출하여 TMO 코드를 생성하는 기능을 수행하며 Java 언어로 구현되어 있다. 최종적 형태의 코드 자동 생성기는 다이어그램 편집기의 애드온 형태로 포함된다. 다이어그램 편집기 상에서 다이어그램 편집이 끝나면 이클립스의 메뉴 창에 있는 코드 생성 버튼을 클릭하는 것으로 자동 생성한다. 하지만 TMO 객체를 사용하기 위해 config.ini 파일이나 TMOSL.dll을 사용하기 위한 프로젝트 속성은 사용자가 직접 수정해야 한다.



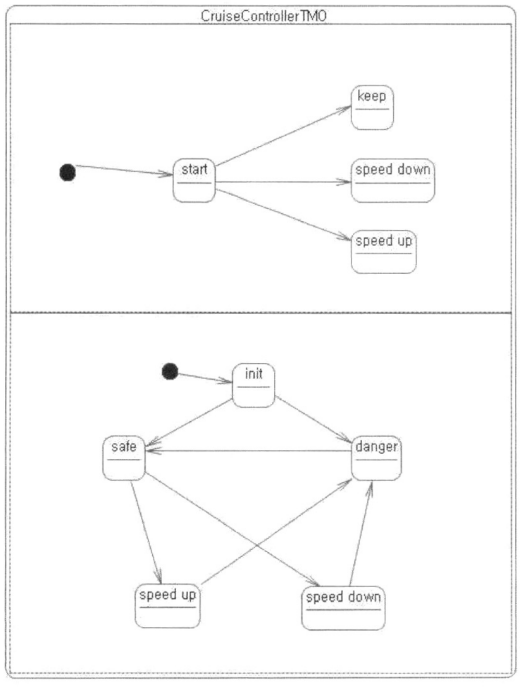
(그림 12) 상태머신 다이어그램에서의 SvM 메소드의 코드 생성 규칙



(그림 13) 순항제어 시스템의 클래스 다이어그램

(그림 13)은 순항제어 시스템의 클래스 다이어그램을 모델링한 것을 보여준다. 현재의 모델 상에서는 시간 제약조건이 보이지 않는데 이것은 위에서 설명한 것과 같이 각 메소드의 속성값으로 입력되었다.

(그림 14)는 CruiseControl_TMO 클래스의 상태 머신 다이어그램을 나타낸다. Distance_TMO와 Speed_TMO 시스템에서 앞차와의 거리와 현재 속도를 측정한 후, Cruise_Controller_TMO 시스템에 전송한다. 이후 Cruise_Controller_TMO 시스템은 측정된 거리와 속도를 바탕으로 가속을 해야 할 경우에는 Accelerate_TMO 시스템을 호출해 가속하고 감속해야 할 경우에는 Break_TMO 시스템을 호출해 감속하는 시스템이다. 트랜지션상에 있는 내용들은 트랜지션상의 속성창에 모두 입력했기에 도구상에 나타나지 않는다.



(그림 14) CruiseControllerTMO 클래스의 상태 머신 다이어그램

(그림 15)는 위의 다이어그램을 입력받아 생성된 예를 보인다.

```
Error Log  Tasks  Problems  Console
Eclipse Application [Eclipse Application] C:\Program Files\Java\jre6\bin\javaw.exe (2010. 11. 29. 오후 4:53:39)
#include <stdio.h>
#include <iostream>
#include "TMOSL.h"
#pragma comment(lib,"TMOSL")
using namespace TMO;
using namespace std;

typedef struct{
} ParamStruct_SvM;

class control_tmo1: public CTMOBase {
private:
        int speed;
        void control_spm(){}{
                ParamStruct_SvM SvMPara;
        }
        int speedup_svm(){}ParamStruct_SvM *para){
        }
        int speeddown_svm(){}ParamStruct_SvM *para){
        }
public:
        //TMO constructor
        control_tmo1(){
                //register control_spm()
                SpM_RegistParam control_spm()_spec;
                AAC aac_control_spm()(_T(""), tm4_DCS_age(0),tm4_DCS_age(0),0, 0, 0, 0);
                control_spm()_spec.build_regist_info_AAC(aac_control_spm());
                RegisterSpM( (PFSpMBody)&control_tmo1::control_spm(), &control_spm()_spec);

                //register speedup_svm()
                SvM_RegistParam speedup_svm()_spec;
                speedup_svm()_spec.GETB =0;
                _tcscpy(speedup_svm()_spec.name,_T("speedup_svm()"));
                RegisterSvM( (PFSvMBody)&control_tmo1::speedup_svm(), &speedup_svm()_spec);

                //register speeddown_svm()
                SvM_RegistParam speeddown_svm()_spec;
                speeddown_svm()_spec.GETB =0;
                _tcscpy(speeddown_svm()_spec.name,_T("speeddown_svm()"));
                RegisterSvM( (PFSvMBody)&control_tmo1::speeddown_svm(), &speeddown_svm()_spec);

                //register TMO
                TMO_RegistParam control_tmo1_spec;
                _tcscpy(control_tmo1_spec.global_name, _T("control_tmo1"));
                control_tmo1_spec.start_time = tm4_DCS_age(2*1000*1000);
                RegisterTMO(&control_tmo1_spec);
        }
};

void main(){
        StartTMOengine();
        control_tmo1 control_tmo1_new;
        MainThrSleep();
}
```

(그림 15) 생성된 코드의 예

6. 결 론

이 연구에서는 분산 실시간 시스템을 지원하기 위해 객체 스스로가 실시간성을 보장하는 TMO 객체기반 코드 자동생성기에 대하여 제안하였다. TMO는 기존의 언어와 달리 시간 제약조건이 구조체로 들어간다는 점과 각 메소드나 클래스를 미들웨어에 등록해야 하는 점, 라이브러리를 상속받아야 하는 점 등이 존재해 기존의 도구로 생성하기에는 어려움이 있었다. 이 연구에서는 TMO 라이브러리를 상속받고

SvM 메소드의 사용에 필요한 게이트를 생성하며 SpM 메소드의 시간제약 조건을 구조체 형태로 자동생성하고 사용을 위해서는 미들웨어에 항상 등록하는 것 등 사용자에게 친숙하지 않은 부분을 자동코드생성기로 지원할 수 있었다.

이 연구에서 개발한 코드 자동생성기는 TMO를 전혀 알지 못해도 UML의 클래스 다이어그램과 상태 머신 다이어그램을 사용할 줄 아는 사용자라면 TMO 코드를 자동으로 생성할 수 있도록 지원하여 사용자에게 편의를 제공하였다. 또한 모델을 통해 사용자의 이해도를 증가시킬 수 있고 이를 통한 유지보수에 시간과 비용을 절감할 수 있는 효과가 기대된다.

참 고 문 헌

[1] Xiaolin Hu, Zeigler, B.P., "Model continuity in the design of dynamic distributed real-time systems," IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, Vol.35, No.6, pp.867-878, 2005.

[2] OMG, Real-time CORBA Specification, version 1.2, 2005.

[3] DREAM Laboratory, University of California, "TMOSL_Manual_v4_2_2," http://dream.eng.uci.edu/TMOdownload/, 2007.

[4] RTAI, the Real time Application Interface for Linux from DIAPM, http://www.rtai.org/

[5] Jung-Guk Kim, Moon Hae Kim, "TMO-eCos: An eCos-based Real-time Micro Operating System Supporting Execution of a TMO Structured Program," International Symposium on Object-Oriented Real-Time Distributed Computing, 2005.

[6] Douglass, Bruce Powel, "실시간 UML", 3판, Addison-Wesley, 2004.

[7] D. Regep, F. Kordon, "Using MetaScribe to prototype a UML to C++/Ada95 code generator," 11th International Workshop on Rapid System Prototyping, pp.128-133, 2000.

[8] M. Usman, A.Nadeem, Tai-hoon Kim, "UJECTOR: A Tool for Executable Code Generation from UML Models," 2008 Advanced Software Engineering and Its Applications, pp.165-170, 2008.

[9] T.G. Moreira, M.A. Wehrmeister, C.E. Pereira, J.-F. Petin, E. Levrat, "Automatic code generation for embedded systems: From UML specifications to VHDL code," 8th IEEE International Conference on Industrial Informatics (INDIN), pp.1085-1090, 2010.

[10] L.B. Brisolara, M.F.S. Oliveira, R. Redin, L.C. Lamb, F. Wagner, "Using UML as Front-end for Heterogeneous Software Code Generation Strategies," Design, Automation and Test in Europe, pp.504-509, 2008.

[11] Rational Rhapsody, http://www.ibm.com/software/awdtools/rhapsody/

[12] L.carnevali, D.D'Amico, L.Ridi, E.Vicario, "Automatic Code Generation from Real-Time Systems Specifications," International Symposium on Rapid System Prototyping, 2009.

[13] G.Bucci, L.Sassoli, E.vicario, "ORIS: a tool for state-space analysis of real-time preemptive systems," International Conference on the Quantitative Evaluation of Systems, 2004.

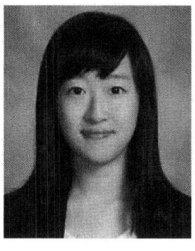
[14] OMG, Unified Modeling Language: Superstructure, version 2.1.1, 2007.

[15] OMG, UML Profile for Schedulability, Performance, and Time Specification, 2002.

[16] Eclipse.org Home, www.eclipse.org

[17] Eclipse, Graphical Modeling Framework, www.eclipse.org/gmf

[18] 박인수, 이정선, 조성래, 정우영, 이우진, "AUTOSAR 기반 차량용 소프트웨어 컴포넌트 모델링 도구", 정보처리학회논문지 A, 제17-A권, 제4호, pp.203-212, 2010년 8월.



석 미 희

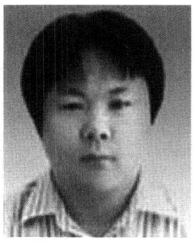
e-mail : miheui.seok@lge.com

2009년 경북대학교 신소재공학과(학사)

2011년 경북대학교 전자전기컴퓨터학부 (공학석사)

2012년~현 재 LG전자 HA사업부 제어 연구소 연구원

관심분야 : 임베디드 소프트웨어 모델링 및 분석, 소프트웨어 테스팅



류 호 동

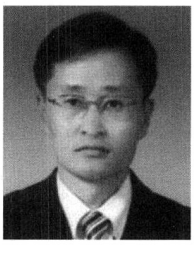
e-mail : hodong@gmail.com

2007년 경북대학교 전자전기컴퓨터학부 (학사)

2009년 경북대학교 전자전기컴퓨터학부 (공학석사)

2009년~현 재 경북대학교 전자전기 컴퓨터학부 박사과정

관심분야 : 컴포넌트 기반 개발방법, 모델기반 코드 자동생성



이 우 진

e-mail : woojin@knu.ac.kr

1992년 경북대학교 컴퓨터과학과(학사)

1994년 한국과학기술원 전산학과 (공학석사)

1999년 한국과학기술원 전산학과 (공학박사)

1999년~2002년 한국전자통신연구원 S/W공학연구부 선임연구원

2002년~현 재 경북대학교 IT대학 컴퓨터학부 부교수

관심분야 : 임베디드 소프트웨어 모델링 및 분석, 정형적 방법, 임베디드 소프트웨어 테스팅 등