TMO 모델 명세 언어 C++T의 설계 및 C++T-to-C++ 번역기의 개발

김 문 회[†] · 박 용 우[††]

요 약

현재, 컴퓨터를 사용하는 실시간 시스템의 응용분야에 대한 요구가 급속히 증가하고 있다. 그러나 이러한 실시간 시스템의 개발은 체계적인 접근방법이 부족하다는 이유로 더디게 진행되고 있다. 체계적인 접근방법의 하나는 실시간 시스템의 시간적인 특성과 기능적인 행동을 잘 추상화 해 줄 수 있는 좋은 모델을 사용하는 것이다. 지금까지 객체지향 개념에 기반한 많은 모델들이 개발되어 소개되고 있다. 그 중 UCI에서 개발된 TMO (Time-triggered Message-triggered Object) 모델은 위에서 언급한 요구사항을 잘 만족시켜 준다.[1] 본 논문에서는 이러한 TMO 모델을 가장 잘 명세해 줄 수 있도록 해 주는 TMO 모델 명세언어인 C++T를 설계하고 이에 대해 상세히 기술한다. 또한, 이러한 C++T 명세를 기존의 언어인 C++로 번역해 줄 수 있는 C++T 번역기를 설계하고 개발한다. 마지막으로, TMO 명세언어 C++T를 기반으로 한 TMO 통합개발환경에 대하여 소개한다.

Design of the TMO Specification Language C++T and Development of the C++T-to-C++ Translator

Moon Hae Kim[†] · Yong Woo Park[††]

ABSTRACT

Currently, application areas of real-time systems using computers are rapidly increasing. However, development of real-time systems is slow because of the lack of a systematic approach. The heart of the systematic approach is an appropriate model for real-time systems which can abstract temporal and functional behaviors of real-time systems. Many models have been proposed. Among them, the TMO(Time-triggered Message-triggered Object) model developed at UCI satisfies above-mentioned requirements.[1] In this paper, we present the design of the C++T language that can specify the TMO model. We have also developed the translator which converts C++T specifications into equivalent C++ codes. We describe the design and implementation of the C++T-to-C++ translator in this paper. Finally, we conclude by describing additional thoughts on the C++T specification language.

1. 서 론

실시간 시스템이 실제로 동작하기 전에 시간 제약 조건을 만족시켜줄 수 있다는 것과 높은 신뢰성을 보장할 수 있다는 것은 미리 검증되어져야 한다. 이러한 실시간 시스템의 개발을 위해서는 위의 특성들이 보장될 수 있다는 것을 검증할 수 있는 체계적인 접근방법과 정확한 지원 도구가 필수적이다. 또한 실시간

※이 연구는 94년도 한국과학재단 연구비 지원에 의한 결과임 (과제번호:941-0900-013-2).

† 정 회 원:건국대학교 컴퓨터공학과
†† 준 회 원:건국대학교 컴퓨터공학과

논문접수:1997년 12월 12일, 심사완료:1997년 1월 16일

시스템의 기능적인 행동이나 시간적인 행동들을 잘 추상화 시켜줄 수 있는 좋은 모델을 사용하는 것이다.

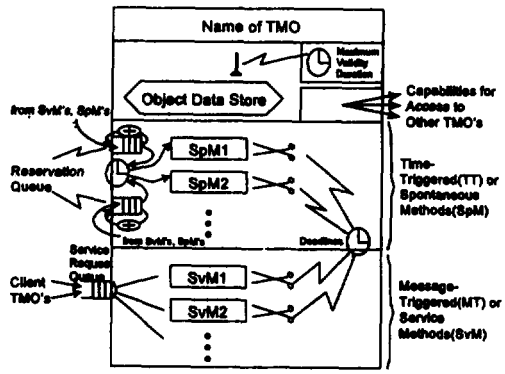
현재 timed petri-net, finite state machine, object-orientation 등에 기반한 다양한 모델들이 소개되고 있는데, 이러한 대부분의 모델들은 실시간 시스템의 시간적인 행동만을 고려하고 있다. 그러나 실시간 시스템이 가지는 시간적인 행동뿐만 아니라 메시지에 의한 기능적인 행동에 대한 추상화 등을 지원해 주기 위해 객체지향 개념에 기반한 TMO 모델이 Kim 등에 의해서 개발되었다.[1, 2]

TMO 모델은 실시간 시스템의 시간적인 행동을 추상화하기 위한 SpM(Spontaneous Method), 메시지에 의한 기능적인 행동을 추상화하기 위한 SvM(Service Method), 그리고 이들이 공유하는 데이터를 세그먼트 단위로 저장하기 위한 ODSS(Object Data Store Segment) 등이 있고 이들을 하나의 객체로 모델링할 수 있도록 해 준다. 이러한 TMO 모델은 실시간 시스템의 디자이너에게 설계 단계에서부터 실시간성을 제공할 수 있도록 해 주며, 실시간 시스템의 추상화에 대한 단순성을 제공해 준다. 그리고 실시간 시스템의 시간적인 분석을 쉽고 간단하게 해 준다.

그러나 TMO 모델을 구현하기 위한 C++ TMO 프로그래밍은 TMO 모델의 정확한 표현이 불가능하므로 TMO 모델이 가지는 실시간성의 보장은 물론 모델의 실시간적인 요소를 알아보기 어렵게 한다. 그리고 기계적으로 기술해 주어야 하는 초기화 코드(dummy code)들 때문에 시간적인 소모가 많고 TMO 모델의 가장 큰 특징인 단순성을 잃게 된다. 따라서 본 논문에서는 기존의 C++ 언어를 확장하여 TMO 모델의 단순성과 실시간성을 그대로 유지하면서 프로그래밍 할 수 있도록 해 주는 새로운 언어인 C++T(C++ Time) 언어를 개발하고, 이를 C++ 언어로 번역해 줄 수 있는 도구인 C++T-to-C++ 번역기(이하 C++T 번역기)를 개발하였다.

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 실시간 시스템에서의 실시간적인 요소들을 실시간 객체로 모델링 하기 위한 기법 중, 실시간적인 특징을 가장 잘 표현 할 수 있는 TMO 모델에 대해서 알아본다. 3장에서는 TMO 모델과 C++ 언어와의 관계를 알아보기 위해 C++ TMO 프로그래밍 예제를 기술하고, 이를 토대로 4장에서는 C++ TMO 프로그래밍에 대한 문제점과 이를 해결하기 위하여 C++ 언어를 확장한 C++T 언어에 관하여 기술한다. 5장에서는 C++T 언어를 C++ 언어로 번역해 주기 위한 C++T 번역기의 설계 및 구현에 관하여 기술한다. 앞에서 기술된 내용을 바탕으로 6장에서는 간단한 C++T TMO 프로그래밍 예를 보여주고, 마지막으로 7장에서는 지금까지 살펴 본 내용을 정리하고 향후 연구 방향을 제시한다.

2. TMO 모델



(그림 1) TMO 모델의 기본적인 구조 [3]
(Fig. 1) Basic structure of the TMO model (adapted from [3])

2.1 TMO 모델의 기본적인 특징

(그림 1)에서 자세히 보여주고 있듯이, TMO 모델만의 두드러지고 고유한 특징은 다음과 같이 크게 두 가지로 나타낼 수 있다.

(a)객체에 포함된 메소드는 크게 두 그룹으로 구분된다. 먼저, 객체의 설계 시에 명세한 시간이나 주기가 되면 실시간 클럭에 의해 실행되는 SpM이 있는데, 이를 time-triggered(TT-) method라 부르기도 한다. SpM의 실행 시작 시간은 설계 단계에서 반드시 상수로 명세되어져야 한다. 이러한 실시간 상수는 SpM 명세의 첫번째 절에 나타나며, 이를 AAC(Autonomous Activation Condition)라 한다. 두 번째는 클라이언트로부터 온 메시지에 의해서 수행되는 SvM이 있는데, 이를 message-triggered(MT-) method라 부르기도 한다. 이러한 TMO 모델의 설계 개념 중 두드러진 특징

은 "RTCS(Real-Time Computing System)는 항상 TMO들로 구성된 네트워크의 형태를 취한다."는 것이다. TMO들은 서버에 있는 SvM에 대한 클라이언트의 호출을 통해서 서로 상호작용 한다. 호출자는 클라이언트 객체에 있는 SpM일 수도 있고, SvM일 수도 있다. 클라이언트는 기본적으로 blocking(or synchronous) call을 통해서 호출하거나, 서버 객체와의 동시 실행을 용이하게 하기 위해서 non-blocking(or asynchronous) call을 사용할 수도 있다.[1]

(b) BCC(Basic Concurrency Constraint)는 TMO들의 시간적인 서비스 능력을 보장하기 위한 제약 조건으로, SpM과 SvM 사이의 충돌을 방지하기 위한 수행 규칙이다. 외부의 클라이언트로부터 온 메시지에 의해 수행되는 SvM의 실행은 기본적으로 SpM과 충돌이 없는 경우나, 충돌이 일어난 SpM의 실행이 끝난 후에만 가능하다. 정확히 말하자면 SpM과 SvM 사이에는 데이터를 공유하기 위한 ODSS가 존재하는데, 이를 동시에 접근하려는 경우에 SpM이 SvM 보다 더 높은 우선순위를 가지는 것이다. 그러므로 객체의 수행되는 시간을 설계 단계에서부터 고정시킬 수 있고, SpM의 수행은 SvM에 의해서 방해 받지 않으며, SpM의 수행 시 그 시간을 보장할 수 있는 것이다.[1]

위의 두 가지 특징 뿐만 아니라, 다음과 같은 기존의 객체 모델에는 나타나지 않는 특징들을 살펴볼 수 있다.

(c) TMO에 나타난 모든 메소드는 데드라인을 갖는다.

(d) TMO에 포함된 실시간 데이터는 유효기간(Maximum Validity Duration)이 지나면 쓸모가 없게 된다.

TMO의 디자이너는 SvM의 명세 시에 데드라인을 나타내줌으로써, 클라이언트 객체의 디자이너에게 시간 서비스 능력에 대한 보장을 제공해 줄 수 있다.[1]

2.2 TMO 모델의 구성요소

TMO 모델은 기존의 객체 모델에 대한 확장으로, 기본적인 구조는 (그림 1)에 잘 나타나 있다. TMO 모델은 시간적인 분석을 용이하게 해 주는 TAC(TMO Access Capability)과 실질적인 구성요소인 ODSS, SpM, 그리고 SvM 등 네 가지의 요소로 구성된다. SpM과 SvM은 앞 절에서 언급한 바와 같고, TAC과 ODSS에 대한 기술은 다음과 같다.

- TAC:어떤 TMO의 SpM이나 SvM에서 다른 TMO에 있는 SvM에 대한 호출이 있을 경우, 이를 위한 통신 채널(Communication Channel)을 할당 받아 가지고 있어야 한다. 채널을 할당 받기 위해서는 호출하려는 SvM의 이름과 이 SvM이 속한 TMO의 이름 등이 필요하다. TAC에 포함되는 정보에는 메시지를 주고 받기 위한 채널의 ID와 데이터를 주고 받기 위한 저장 장소 등이 있다.
- ODSS:어떤 TMO 모델 내의 SpM과 SvM이 서로 공유하는 데이터를 저장하기 위한 공간인 ODS(Object Data Store)가 있는데, 이 ODS는 세그먼트 단위로 공유된다. 이를 ODSS라 한다. 그리고 이러한 데이터는 유효기간이 지나면 무효한 데이터가 된다.

이렇게 TMO를 구성하는 모든 구성요소들이 가지는 시간적인 특성을 설계 단계에서부터 정확히 명시해 줌으로써, 그 TMO에 대한 시간적인 특성의 분석을 용이하게 해 주고, 시스템의 설계 단계에서부터 시스템에 대한 실시간성을 보장해 준다.

3. TMO 모델과 C ++ TMO 프로그래밍

이 장에서는 TMO 모델과 이를 C++ 언어를 이용하여 프로그래밍하는 방법에 대해 살펴보기 위해 먼저 가상의 시나리오를 설정하여, 이를 TMO 모델로 모델링하고, 이 TMO 모델을 C++ 언어를 이용하여 프로그래밍 한다. 또한 이러한 C++ TMO 프로그래밍 시에 발생하는 문제점을 살펴본다.

3.1 시나리오

시나리오는 다음과 같은 역할을 하는 두 개의 태스크로 구성된다.

① 정수형 값을 주기적으로 생성하여 가지고 있다.

② 위의 태스크에서 정수형 값을 주기적으로 가져와 이를 디스플레이 한다.

먼저 정수형 값을 주기적으로 생성하여 저장하고 있는 태스크를 정수 생성자 태스크(Integer Generator Task)라 한다. 이 태스크는 정수형 값을 주기적으로 생성하는 주기성을 가지며, 또한 다른 태스크로부터 정수 값에 대한 요청을 받아 이 값을 전달해 주는 서

비스를 수행하여야 한다.

두 번째 태스크는 정수 요청자 태스크(Integer Requester Task)라 하고, 이 태스크의 역할은 첫번째 태스크에게 주기적으로 정수 값을 요청해야 하고, 이 얻어진 정수 값을 화면상에 디스플레이 한다.

위에서 설명한 두 개의 태스크를 TMO로 모델링하고, 이를 실제로 구현하기 위해 C++ 언어를 이용한 C++ TMO 프로그래밍 방법과 그 때의 문제점들에 대해서 살펴본다.

3.2 TMO 모델링

TMO 모델은 2장에서 자세히 살펴보았듯이, 크게 3개의 주요 구성요소와 1개의 추가 구성요소로 나눌 수 있다. 먼저 주요 구성요소에는 TMO 내의 두 개의 메소드 그룹인 SpM 및 SvM과 이들이 공유하는 데이터를 저장하기 위한 ODSS가 있고, 추가 구성요소에는 TMO들간의 호출을 위한 정보를 가지는 TAC이 있다. 3.1 절의 시나리오에 나타난 두 개의 태스크를 각각 TMO 모델로 모델링하여 보면 다음과 같다.

3.2.1 정수 생성자 TMO

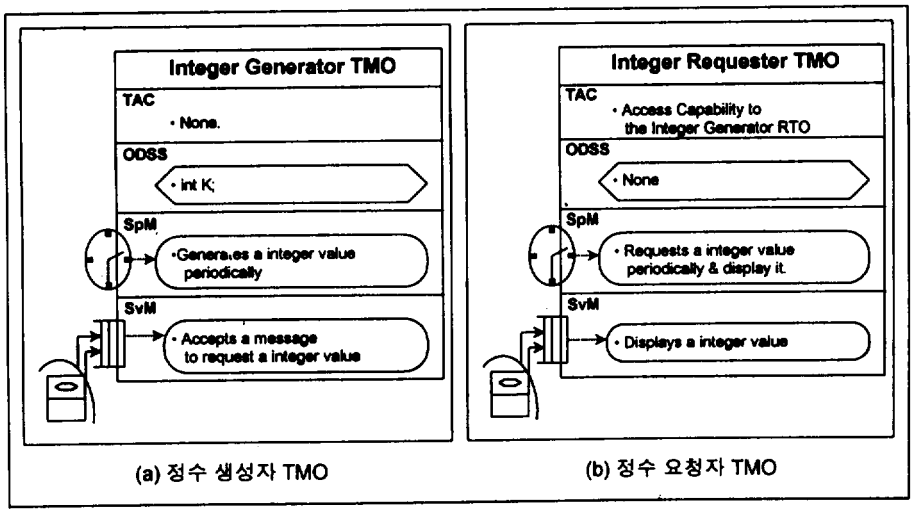
정수 생성자 태스크의 역할은 먼저 주기적으로 새로운 정수 값을 생성하여 저장하고 있으면서, 다른 태스크로(정수 요청자 태스크)부터 온 요청을 받아 저장하고 있는 정수 값을 전달해 주는 서비스를 수행해야 하는데, TMO 모델로 모델링 된 정수 생성자 태스크를 정수 생성자 TMO라 부르기로 한다.

(그림 2)의 (a)에 나타나 있는 정수 생성자 TMO를 세부적으로 살펴보면 다음과 같다.

- TAC: 다른 TMO에 있는 SvM을 호출하지 않으므로 TAC은 필요없다.
- ODSS: SpM에서 주기적으로 정수 값을 생성하여 그 값을 저장하고 있어야 하므로 이를 위한 정수형 변수 'K'를 필요로 하고, SvM에서 서비스 요청이 있을 때 읽어야 하므로 SpM과 SvM이 데이터를 공유하기 위한 ODS에 선언되어야 한다.
- SpM: 주기적으로 정수 값을 생성한다.
- SvM: 정수 값에 대한 요청을 받아, 그 정수 값을 건네주는 서비스를 수행한다.

3.2.2 정수 요청자 TMO

정수 요청자 태스크의 역할은 먼저 정수 생성자 태스크에 정수 값을 주기적으로 요청하여 받아 오고, 이 정수 값을 디스플레이 해 주는 서비스를 수행하는데, TMO 모델로 모델링 된 정수 요청자 태스크를 정



(그림 2) 정수 생성자 TMO와 정수 요청자 TMO
(Fig. 2) Integer generator TMO and integer requester TMO

수 요청자 TMO라 한다. (그림 2)의 (b)에 나타나 있는 정수 요청자 TMO를 세부적으로 살펴보면 다음과 같다.

- TAC:정수 생성자 TMO의 SvM을 주기적으로 호출하여 정수 값을 가져와야 하므로 정수 생성자 TMO의 해당 SvM에 대한 TAC을 갖는다.
- ODSS:SpM과 SvM 사이에 특별히 공유할 데이터는 없다.
- SpM:정수 생성자 TMO의 SvM을 주기적으로

```
/* TAC 섹션 */

/* ODSS 섹션 */
class IntegerGeneratorODSSClass
      : public BasicODSSClass
{
  private:
    int K;
  public:
    IntegerGeneratorODSSClass(void);
   ~IntegerGeneratorODSSClass(void);
    int get_value(void);
    void set_value(int);
};

/* SpM 섹션 */
class IntegerGeneratorSpM1Class
      : public BasicSpMClass
{
  private:
    IntegerGeneratorODSSClass *ODSS;
    void SpM1Body(void);
  public:
    IntegerGeneratorSpM1Class
      (IntegerGeneratorODSSClass *);
};

/* SvM 섹션 */
class IntegerGeneratorSvM1Class
      : public BasicSvMClass
{
  private:
    IntegerGeneratorODSSClass *ODSS;
    void SvM1Body(void);
  public:
    IntegerGeneratorSvM1Class
      (IntegerGeneratorODSSClass *);
};

/* TMO 섹션 */
class IntegerGeneratorTMOClass
{
  private:
    IntegerGeneratorODSSClass ODSS;
  public:
    IntegerGeneratorSpM1Class *SpM1;
    IntegerGeneratorSvM1Class *SvM1;

    IntegerGeneratorTMOClass() {
      new(SpM1, IntegerGeneratorSpM1Class(&ODSS));
      new(SvM1, IntegerGeneratorSvM1Class(&ODSS));
    }
};
```

(a) 정수 생성자 TMO

```
/* TAC 섹션 */
class IntegerRequesterTAC
      : public BasicRTOAccessCapabilityClass
{
public:
    int DFCID_IntegerGeneratorSvM1;
    IntegerGeneratorSvM1_Par_t
IntegerGeneratorSvM1_Par;

    void Register(void) {
      RegisterRTOAccessCapability(
        "IntegerGeneratorTMO",
        "SvM1", &DFCID_IntegerGeneratorSvM1, 1000);
    }
};

/* ODSS 섹션 */
/* SpM 섹션 */
class IntegerRequesterSpM1Class
      : public BasicSpMClass
{
  private:
    IntegerRequesterTAC *TAC;
    void SpM1Body(void);
  public:
    IntegerRequesterSpM1Class
      (IntegerRequesterTAC *);
};

/* SvM 섹션 */
class IntegerRequesterSvM1Class
      : public BasicSvMClass
{
  private:
    void SvM1Body(void);
  public:
    IntegerRequesterSvM1Class();
};

/* TMO 섹션 */
class IntegerRequesterTMOClass
{
  private:
    IntegerRequesterTAC TAC;
  public:
    IntegerRequesterSpM1Class *SpM1;
    IntegerRequesterSvM1Class *SvM1;
    IntegerRequesterTMOClass() {
      new(SpM1, IntegerRequesterSpM1Class(&TAC));
      new(SvM1, IntegerRequesterSvM1Class());
    }
    void RegisterTAC(void) { TAC.Register(); }
};
```

(b) 정수 요청자 TMO

(그림 3) 정수 생성자 TMO와 정수 요청자 TMO에 대한 C++ TMO 프로그램 헤더
(Fig. 3) Program header for integer generator TMO and integer Requester TMO

호출하여 정수 값을 가져오고, 다시 이 값을 자신(정수 요청자 TMO)의 서비스 메소드를 호출하여 디스플레이 해 주어야 한다.

- SvM : 정수 값의 디스플레이 요청을 받아 서비스해 주는 기능을 갖는다.

정수 생성자 태스크와 정수 요청자 태스크를 TMO 모델로 모델링하여 보았는데, 상당히 정확하고 간단하게 표현되며, 모델링할 때 클라이언트/서버 개념과 객체지향 개념을 충실히 따를 수 있다는 것을 알 수 있다.

3.3 C++ TMO 프로그래밍

현재까지는 TMO 모델을 실질적으로 프로그래밍하기 위해서 C++ 언어를 사용하고 있다. 본 절에서는 정수 생성자 TMO와 정수 요청자 TMO를 C++ 언어를 사용하여 프로그래밍하는 방법을 보여준다.

3.3.1 정수 생성자 TMO에 대한 C++ TMO 프로그래밍

정수 생성자 TMO를 C++로 프로그래밍하기 위해서는 각 구성요소들을 각각의 클래스로 선언한다. 각 구성요소 별로 TAC 섹션, ODSS 섹션, SpM 섹션, SvM 섹션, 그리고 TMO 섹션 등 크게 5개의 섹션으로 나누어 각각을 클래스화 한다. (그림 3)의 (a)와 (그림 4)의 (a)는 정수 생성자 TMO에 대한 C++ TMO 프로그램을 잘 보여주고 있다.

- TAC 섹션 : 없음.
- ODSS 섹션 : 정수 생성자 TMO는 SpM에서 생성된 정수 값을 SvM과 공유해야 하는데, SpM은 이 정수 값의 저장공간에 대해서 RW(Read and Write) 권한을 가지며, SvM은 이 저장 공간에 대해서 R(Read) 권한만 가지면 된다. SpM과 SvM이 ODSS를 접근할 때 충돌이 발생할 경우 이를 해결하기 위해 ODSS에 CREW(Concurrent Read and Exclusive Write) 모니터 기능을 제공해 주며, 다음과 같은 함수들이 제공된다.
  - EnterCREW_RO() or EnterCREW_RW(): CREW 모니터 기능을 설정한다.
  - ExitCREW_RO() or ExitCREW_RW():CREW 모니터 기능을 종료한다.
- SpM 섹션 : 정수 생성자 TMO의 SpM은 주기적으로 임의의 정수를 생성하여, ODSS에 이 정수 값을 저장해 놓는다. 다음은 SpM을 위해 제공되는 함수들이다.
  - RegisterAndInitializeSpMwithBody(SpM1Body): SpM을 등록한다.
  - ReportCompletion() : SpM이 종료되었음을 알려 준다.
- SvM 섹션 : 정수 생성자 TMO의 SvM은 정수 요청자 TMO로부터 온 정수 값의 요청에 대해, 자신이 속한 TMO내의 ODSS에 있는 정수 값을 읽어, 그 값을 건네 주는 서비스를 수행한다. 다음은 SvM을 위해 제공되는 함수들이다.
  - RegisterAndInitializeSvMwithBody(SvM1Body): SvM을 등록한다.
  - ReportCompletion() : SvM의 수행을 종료한다.
  - ReceiveSRQ() : SRQ(Service Request Queue)에서 파라메터를 얻는다.
  - ReplySRQ() : SRQ에 결과 값을 건네준다.
  - ReportCompletion() : SvM이 종료되었음을 알려 준다.
- TMO 섹션 : 각각의 클래스로 선언되어 있는 구성요소들을 하나의 TMO 객체로 정의해 준다.

3.3.2 정수 요청자 TMO에 대한 C++ TMO 프로그래밍

정수 요청자 TMO에 대한 C++ TMO 프로그래밍 방법은 정수 생성자 TMO에 대한 C++ TMO 프로그래밍 방법과 같다. 다만 정수 생성자 TMO에는 포함되어 있지 않은 TAC 섹션에 대한 C++ TMO 프로그램 코드만 살펴본다. (그림 3)의 (b)와 (그림 4)의 (b)는 정수 요청자 TMO에 대한 C++ TMO 프로그램을 잘 보여주고 있다.

- TAC 섹션 : 정수 요청자 TMO는 정수 생성자 TMO의 SvM1을 호출해야 한다. 따라서 정수 생성자 TMO의 SvM1을 호출하기 위한 TAC을 갖는다. 다음은 TAC을 위해 제공되는 함수들이다.
  - RegisterRTOAccessCapability() : TAC을 위한 채널을 할당받는다.

• ODSS 섹션 : 없음.

• SpM 섹션 : 정수 요청자 TMO의 SpM은 주기적으로 정수 생성자 TMO의 SvM1에게 정수 값을 요청하여 가져오고, 이를 화면상에 디스플레이 해 준다. 다음은 SvM을 호출하기 위해 제공되는 함수와 방법이다.

- BlockingSRQ() : SvM에 대한 blocking call을 수행한다.
- NonBlockingSRQ() : SvM에 대한 non-blocking call을 수행한다.
- ClientTransfer() : SvM에 대한 client-transfer call을 수행한다.

```
/* ODSS 섹션 */
int IntegerGeneratorODSSClass::get_value(void)
{
  int i;
  EnterCREW_RO(); i = K; ExitCREWwithValue_RO(i);
}

void IntegerGeneratorODSSClass::set_value(int i)
{
  EnterCREW_RW(); K = i; ExitCREW_RW();
}

/* SpM 섹션 */
IntegerGeneratorSpM1Class::IntegerGeneratorSpM1Class
(IntegerGeneratorODSSClass *odss_ptr)
{
  ODSS = odss_ptr;
  ......
  RegisterAndInitializeSpMwithBody(SpM1Body);
}

void IntegerGeneratorSpM1Class::SpM1Body(void)
{
  ODSS->set_value((int)random(60000));
  ReportCompletion();
}

/* SvM 섹션 */
IntegerGeneratorSvM1Class::IntegerGeneratorSvM1Class
(IntegerGeneratorODSSClass *odss_ptr)
{
  ODSS = odss_ptr;

  strcpy(SvMParameter.RTO_Name,
         "IntegerGeneratorTMO");
  strcpy(SvMParameter.SvM_Name, "SvM1");
  ......
  RegisterAndInitializeSvMwithBody(SvM1Body);
}

void IntegerGeneratorSvM1Class::SvM1Body(void)
{
  int i, timestamp, client_dfcid_rr;

  ReceiveSRQ(&client_dfcid_rr, &i, &timestamp);
  i = ODSS->get_value();
  ReplySRQ(client_dfcid_rr, &i, sizeof(int), timestamp);

  ReportCompletion();
}
```

(a) 정수 생성자 TMO

```
/* ODSS 섹션 */

/* SpM 섹션 */
IntegerRequesterSpM1Class::IntegerRequesterSpM1Class
  (IntegerRequesterTAC *tac_ptr)
{
  TAC = tac_ptr;
  ......
  RegisterAndInitializeSpMwithBody(SpM1Body);
}

void IntegerRequesterSpM1Class::SpM1Body(void)
{
  int i;

  BlockingSRQ(TAC->DFCID_IntegerGeneratorSvM1,
              &i, sizeof(int), 1000); // Deadline

  gotoxy(10, 13);
  printf("%8d", i);

  ReportCompletion();
}

/* SvM 섹션 */
IntegerRequesterSvM1Class::IntegerRequesterSvM1Class()
{
  strcpy(SvMParameter.RTO_Name,
         "IntegerRequesterTMO");
  strcpy(SvMParameter.SvM_Name, "SvM1");
  ......
  RegisterAndInitializeSvMwithBody(SvM1Body);
}

void IntegerRequesterSvM1Class::SvM1Body(void)
{
  int i, timestamp, client_dfcid_rr;

  ReceiveSRQ(&client_dfcid_rr, &i, &timestamp);

  gotoxy(10, 14); printf("%8d", i);

  ReportCompletion();
}
```

(b) 정수 요청자 TMO

(그림 4) 정수 생성자 TMO와 정수 요청자 TMO에 대한 C++ TMO 프로그램 코드

(Fig. 4) C++ TMO program code for integer generator TMO and integer requester TMO

3.3.3 AIP(Application Initial Process) 구현 및 Makefile

정수 생성자 TMO와 정수 요청자 TMO를 실제 프로세스로 만들어 주기 위한 AIP와 C++ TMO 프로그램을 컴파일하기 위한 makefile은 (그림 5)에 자세히 나타나 있다.

다음은 AIP를 위해 제공되고 있는 함수와 클래스이다.

- AIPSupportClass : AIP에서 해야 할 기본적인 역할을 담당하는 클래스이다.
- TMOExecManClass : TMO 실행과 관련된 커널 서비스를 호출하는 C++ TMO 프로그램을 위한 인터페이스를 제공해 준다.
- TMOMan.CreateMEP() : 기본적인 크기의 스택과 PBR(Priority Bracket Record)을 가지는 MEP (Method Execution Process)를 생성한다.
- TMOMan.BindMethodToMEP() : SpM을 위한 MEP을 미리 정적으로 할당해 준다. SvM은 서비스 요청이 있을 때만 수행되므로 MEP을 동적으로 할당해 준다.
- AIP.PermanentSleep() : AIP는 일종의 초기화를 위한 프로세스로써 TMO 프로세스들을 생성해 준 후, 자신의 역할은 모두 수행되었으므로 영원히 잠들고 새로 생성된 TMO 프로세스들(SpM or SvM)이 수행된다.

각각의 목적 파일들은 'BCC32' 컴파일러를 이용하여 컴파일한다. 그리고 '386link'를 이용하여 위에서 생성된 목적 파일들과 제공되는 라이브러리를 링크한다.

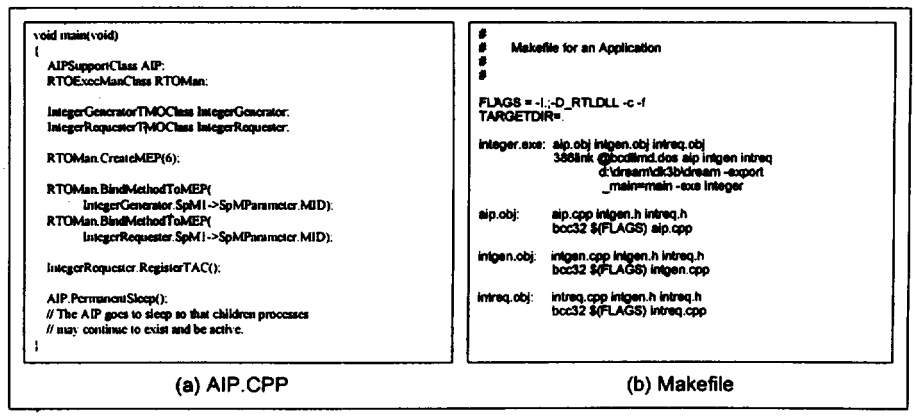
4. TMO 모델 명세 언어 C++T

본 장에서는 C++ TMO 프로그래밍의 장·단점과 이를 보완하기 위한 TMO 모델 명세 언어 C++T의 필요성에 대해서 살펴보고, C++ 언어를 확장하여 TMO 모델 명세 언어 C++T 언어를 정의하여 본다.

4.1 C++ TMO 프로그래밍의 장·단점

3 장에서는 가상으로 설정한 시나리오에 나타난 정수 생성자 TMO와 정수 요청자 TMO를 C++ TMO 프로그래밍하여 보았다. 이를 바탕으로 TMO 모델에 대한 C++ TMO 프로그래밍 시에 발생하는 몇 가지 문제점 및 불편한 점을 다음과 같이 나타낼 수 있다.

- TMO 모델이 가지는 단순성과 정확한 시간 명세 특성을 잃게 된다.
- TMO의 구성요소를 초기화하기 위해 요구되는 추가적인 C++ 코드가 많다.
- TMO 모델의 각 메소드 수행과 관련된 데드라인, ODSS의 유효기간, 그리고 SpM의 시간적인 명세인 AAC 등의 시간적인 요소에 대한 표현이 어



(a) AIP.CPP

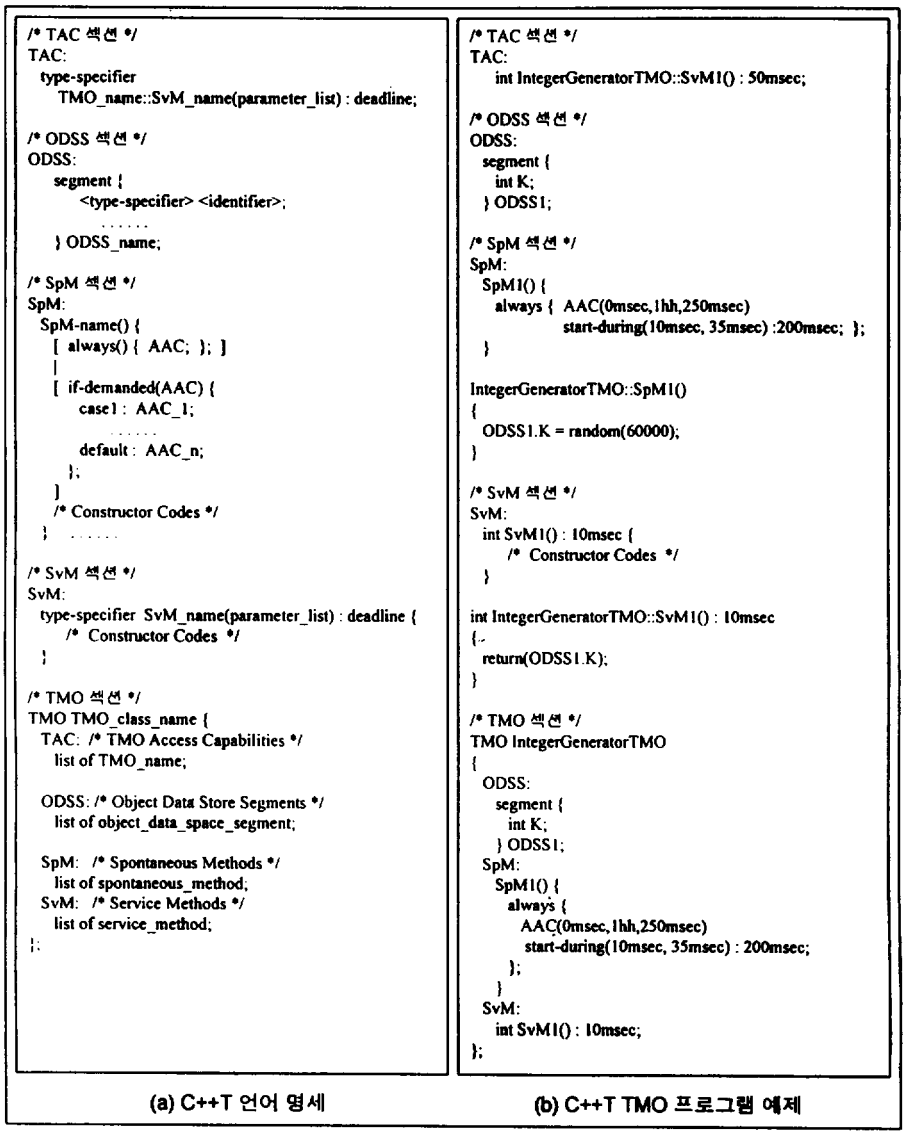
```
void main(void)
{
  AIPSupportClass AIP;
  RTOExecManClass RTOMan;

  IntegerGeneratorTMOClass IntegerGenerator;
  IntegerRequesterTMOClass IntegerRequester;

  RTOMan.CreateMEP(6);

  RTOMan.BindMethodToMEP(
        IntegerGenerator.SpM1->SpMParameter.MID);
  RTOMan.BindMethodToMEP(
        IntegerRequester.SpM1->SpMParameter.MID);

  IntegerRequester.RegisterTAC();

  AIP.PermanentSleep();
  // The AIP goes to sleep so that children processes
  // may continue to exist and be active.
}
```

(b) Makefile

```
#
#     Makefile for an Application
#
#

FLAGS = -I.;-D_RTLDLL -c -f
TARGETDIR=.

integer.exe:  aip.obj intgen.obj intreq.obj
              386link @bcc1lmd.dos aip intgen intreq
                    d:\dream\d63b\dream -export
                    _main=main -exe integer

aip.obj:      aip.cpp intgen.h intreq.h
              bcc32 $(FLAGS) aip.cpp

intgen.obj:   intgen.cpp intgen.h intreq.h
              bcc32 $(FLAGS) intgen.cpp

intreq.obj:   intreq.cpp intgen.h intreq.h
              bcc32 $(FLAGS) intreq.cpp
```

(그림 5) 정수 생성자 TMO와 정수 요청자 TMO를 위한 AIP 및 makefile

(Fig. 5) AIP and makefile for integer generator TMO and integer requester TMO



```
/* TAC 섹션 */
TAC:
  type-specifier
    TMO_name::SvM_name(parameter_list) : deadline;

/* ODSS 섹션 */
ODSS:
    segment {
        <type-specifier> <identifier>;
          .....
    } ODSS_name;

/* SpM 섹션 */
SpM:
  SpM-name() {
    [ always() { AAC; }; ]
    |
    [ if-demanded(AAC) {
        case1 : AAC_1;
          .....
        default : AAC_n;
      };
    ]
    /* Constructor Codes */
  }   .....

/* SvM 섹션 */
SvM:
  type-specifier SvM_name(parameter_list) : deadline {
      /* Constructor Codes */
  }

/* TMO 섹션 */
TMO TMO_class_name {
  TAC: /* TMO Access Capabilities */
    list of TMO_name;

  ODSS: /* Object Data Store Segments */
    list of object_data_space_segment;

  SpM:  /* Spontaneous Methods */
    list of spontaneous_method;
  SvM:  /* Service Methods */
    list of service_method;
};
```

(a) C++T 언어 명세

```
/* TAC 섹션 */
TAC:
    int IntegerGeneratorTMO::SvM1() : 50msec;

/* ODSS 섹션 */
ODSS:
  segment {
    int K;
  } ODSS1;

/* SpM 섹션 */
SpM:
  SpM1() {
    always { AAC(0msec,1hh,250msec)
             start-during(10msec, 35msec) :200msec; };
  }

IntegerGeneratorTMO::SpM1()
{
  ODSS1.K = random(60000);
}

/* SvM 섹션 */
SvM:
  int SvM1() : 10msec {
      /* Constructor Codes */
  }

int IntegerGeneratorTMO::SvM1() : 10msec
{
  return(ODSS1.K);
}

/* TMO 섹션 */
TMO IntegerGeneratorTMO
{
  ODSS:
    segment {
      int K;
    } ODSS1;
  SpM:
    SpM1() {
      always {
        AAC(0msec,1hh,250msec)
        start-during(10msec, 35msec) : 200msec;
      };
    }
  SvM:
    int SvM1() : 10msec;
};
```

(b) C++T TMO 프로그램 예제

(그림 6) C++T 언어 명세 및 C++T TMO 프로그램 예제
(Fig. 6) Specification written in C++T language and example of C++T TMO program

렵고 복잡하다.

- C++ 언어에서 시간적인 표현이 복잡하고 어려우므로 C++ TMO 프로그램에 대한 시간적인 분석이 어렵고 복잡하다.

위에서 나열한 단점에도 불구하고, C++ 언어만이 가지는 장점을 다음과 같이 나타낼 수 있다.

- 강력한 객체 지향적인 특성을 갖는다.
- 전처리기 및 컴파일러와 같은 강력한 도구를 재사용할 수 있다.
- TMO 모델을 위해 제공되고 있는 라이브러리가 C++ 클래스로 구현되어 있다.

4.2 TMO 모델 명세 언어 C++T의 필요성 및 고려 사항

TMO 모델에 대한 C++ TMO 프로그래밍의 장·단점을 보완하고, TMO 모델의 디자이너 및 프로그래머에게 주어진 노력을 최소화할 수 있도록 하기 위해 TMO 모델을 위한 명세 언어 C++T의 개발이 필요하다. 그러나 이러한 C++T를 개발할 때 다음과 같은 사항을 고려해야 한다.

- TMO 모델이 가지는 단순성과 정확한 시간 명세 특성을 그대로 유지한다.
- 시간적인 표현에 있어 단순성을 제공한다.
- 설계 단계에서 보장하는 실시간성을 구현 단계에서도 보장할 수 있어야 한다.
- C++ 사용자들에게 새로운 언어에 대한 거부감을 주지 않는다.
- C++ 언어의 객체 지향적인 특성을 그대로 유지한다.
- C++ 언어에서 사용하는 전처리기와 컴파일러를 그대로 사용한다.
- 프로그래머에게 TMO 모델에 대한 투명성을 제공한다.

4.3 TMO 모델 명세 언어 C++T

TMO 모델 명세 언어 C++T를 설계할 때, 먼저 TMO 모델의 각 구성요소에 대한 명세를 가능하게 하여야 한다. 따라서 본 절에서는 이 C++T 언어를 이용하여 3장에서 소개한 정수 생성자 TMO와 정수 요청자 TMO에 대한 C++T TMO 프로그래밍 하여 봄으로써 C++T 언어에 대하여 살펴본다. (그림 6)은 C++T 언어 명세와 그 예제를 자세히 보여주고 있다.

- TAC 섹션:TMO 모델의 ODSS는 유효기간(MVD), SpM은 AAC, 그리고 SvM은 데드라인을 각각 갖는다. 따라서 프로그램 전체 코드를 스캔하면서 어떤 TMO 내에서의 SvM에 대한 모든 호출 관계를 알 수 있으므로 굳이 TAC을 명세할 필요는 없다. 그러나 다른 TMO의 어떤 SvM을 호출할 때, 실시간적인 시간 분석을 위해 TAC 섹션 내에서 해당 SvM에 대한 데드라인 등을 명시해 줌으로써 시간 분석을 용이하게 할 수 있다.
- ODSS 섹션:유효기간은 다른 데드라인 표현과 같이 데이터의 선언 다음에 ':MVD;' 추가함으로써 명세할 수 있도록 한다. ODSS는 세그먼트 단위로 공유되므로 ODSS 섹션에 선언된 변수들은 세그먼트 단위로 선언되어지도록 한다. 그리고 데이터에 대한 시간적인 표현을 위해서 다음과 같은 새로운 데이터 형을 기본적으로 제공한다.
  - time type : time = {hh : mm : ss.msec, default = msec};
  - deadline : time type

  그리고 C++T 번역기는 ODSS 섹션을 위해 다음과 같은 코드를 추가로 생성해 줌으로써, 프로그래머에게 CREW 모니터에 대한 투명성을 제공해 주어야 한다.
  - EnterCREW_RO() or EnterCREW_RW()
  - ExitCREW_RO(), or ExitCREW_RW()
  - ExitCREWwithValue_RO() or ExitCREWwithValue_RW()
- SpM 섹션:SpM의 수행은 AAC절이 always일 경우는 always 다음에 나오는 AAC절에 의해 정적(static)으로 정해지며, if-demanded일 경우는 그 SpM을 호출할 때, 파라메터를 이용하여 해당 AAC를 동적(dynamic)으로 지정해 줄 수 있으며, 그에 해당하는 AAC절에 나타난 대로 수행된다. 만약 SpM의 파라메터가 없다면 always 절이 나타나야만 한다. Always와 if-demanded를 위한 명세는 C++ error(or exception) handler 내부의 'try' 및 'catch'를 위한 명세와 같은 형태를 취한다. SpM에 대한 C++T 명세를 정의할 때, 이는 TMO 클래스의 멤버함수로써 정의되어야 한다. 왜냐하면 SpM은 TMO의 메소드이기 때문이다.
- SvM 섹션:SvM에 대한 C++T 명세가 정의될 때, 이는 TMO 클래스의 멤버함수로써 정의되어야 한다. SvM에 대한 input은 파라메터를 통해서 전달하고, output은 리턴 값을 통해서 전달되어 C++ 언어에서의 함수와 같게 선언되어져야 한다. 그리고 SvM에 대한 다음과 같은 형식의 호출에 대해서도 지원해 주어야 한다.
  - BLC : BlockingCall()
  - NBC : NoneBlockingCall()
  - CTF : ClientTransfer()

이러한 호출관계에 대해서는 SvM에 대한 C++T 명세가 정의될 때, 유닉스 시스템에서 시스템 호출 시 사용되는 mode 파라메터를 추가함으로써 이를 해결한다.

- TMO 섹션:TMO 모델의 모든 구성요소들은 TMO 클래스 내에 선언함으로써 TMO를 나타내게 할 수 있다. C++ 언어에는 없는 TMO를 위한 새로운 클래스인 TMO 클래스에 대한 명세를 가능하게 하여 TMO를 단순하게 하나의 객체로 표현 가능하도록 한다.

5. C++T 번역기의 설계 및 개발

TMO 모델에 대한 C++T TMO 프로그램을 기존

```
/* C++T 토큰 분석기의 Rules section */
%%
                .....

TMO         { TMO_flag = FLAG_ON;
              TMO_filter(yytext, TMO); }
TAC         { TMO_filter(yytext, TAC); }
ODSS        { TMO_filter(yytext, ODSS); }
SpM         { TMO_filter(yytext, SPM); }
SvM         { TMO_filter(yytext, SVM); }
segment     { TMO_filter(yytext, SEGMENT); }
always      { TMO_filter(yytext, ALWAYS); }
if-demanded { TMO_filter(yytext, IF_DEMANDED); }
AAC         { TMO_filter(yytext, AAC); }.
start-during { TMO_filter(yytext, START_DURING);
}
hh          { TMO_filter(yytext, HH); }
mm          { TMO_filter(yytext, MM); }
ss          { TMO_filter(yytext, SS); }
msec        { TMO_filter(yytext, MSEC); }
BLC         { TMO_filter(yytext, BLC); }
NBC         { TMO_filter(yytext, NBC); }
CTF         { TMO_filter(yytext, CTF); }
"{"         { if(TMO_flag >= FLAG_ON)
                TMO_flag++; TMO_filter(yytext, '{');
}
"}"         { if(TMO_flag >= FLAG_ON) {
                if(--TMO_flag == FLAG_ON)
                  TMO_flag = FLAG_OFF;
                TMO_filter(yytext, '}'); }

%%
```

(a) C++T 토큰 분석기

```
/* C++T 문법 */

TMO_member_declaration_list_opt:   /* nothing */
  | TMO_member_declaration_list_opt
    TMO_member_declaration
  ;
TMO_member_declaration:
    TAC ':'    TAC_declaration_list_opt
  | ODSS ':'   ODSS_declaration_list_opt
  | SPM ':'    SpM_declaration_list_opt
  | SVM ':'    SvM_declaration_list_opt
  ;
TAC_declaration_list_opt:    /* nothing */
  | TAC_declaration_list_opt TAC_member_declaration
  ;
TAC_member_declaration:
    type_name IDENTIFIER
              CLCL IDENTIFIER parameter_type_list ';'
  ;
ODSS_declaration_list_opt:   /* nothing */
  | ODSS_declaration_list_opt ODSS_member_declaration
  ;
ODSS_member_declaration:
    SEGMENT '(' declaration_list ')' IDENTIFIER ';'
  | declaration_list
  ;
SpM_declaration:
    IDENTIFIER '(' ')' '{' ALWAYS '(' ')'
        AAC_specifier '}' ';'
      statement_list_opt
    '}'
  | IDENTIFIER '(' AAC ')' '{'
      IF_DEMANDED '(' AAC ')' '{'
        AAC_specifier_switch_list '}' ';'
      statement_list_opt
    '}'
  ;
AAC_specifier_switch_list:
    AAC_specifier_switch
  | AAC_specifier_switch_list AAC_specifier_switch
  ;
AAC_specifier_switch:
    CASE IDENTIFIER ':' AAC_specifier
  | DEFAULT ':' AAC_specifier
  ;
AAC_specifier:
    AAC '(' TMO_aggregate_time_type ','
        TMO_aggregate_time_type ','
        TMO_aggregate_time_type ')'
  | START_DURING '('
        TMO_aggregate_time_type ','
        TMO_aggregate_time_type ')'
        TMO_deadline_opt ';'
  ;
SvM_declaration_list_opt:    /* nothing */
  | SvM_declaration_list_opt SvM_declaration
  ;
SvM_declaration:
    SvM_head_declaration ';'
  | SvM_head_declaration compound_statement
  ;
SvM_head_declaration:
    type_name IDENTIFIER parameter_type_list
  ;
```

(b) C++T 문법

(그림 7) C++T 토큰 분석기 및 문법
(Fig. 7) C++T token analyzer and syntax

의 C++ 전처리기와 컴파일러를 이용하기 위해 C++ 코드로 번역해 줄 수 있는 C++T 번역기가 필요하다. C++T 번역기는 기존의 C++ 언어를 위한 LEX 코드와 YACC 문법을 확장하는 방식으로, C++ 언어의 LEX 코드에 TMO 모델 관련 토큰을 추가하고, YACC 문법에 TMO 모델 관련 문법을 추가하여 이를 바탕으로 C++T 번역기를 개발하였다.

5.1 C++T-to-C++ 토큰 분석기의 설계 및 개발

C++T-to-C++ 토큰 분석기(이하 C++T 토큰 분석기)를 위해 기존의 C++ 토큰 분석기를 확장하고, 이를 Flex를 이용하여 토큰 분석기 코드를 생성하였다. C++T 토큰 분석기의 가장 중요한 역할은 TMO 모델 관련 토큰들과 기존의 토큰들을 나누어 주는 것이다. C++T의 토큰 분석기는 TMO 클래스 내부의 모든 토큰들과 TMO 멤버함수 내의 모든 토큰들을 C++T 파서에게 넘겨주어 파서가 처리하게 하고, 그 이외의 토큰들에 대해서는 토큰 분석기에서 처리하게 한다. (그림 7)의 (a)는 C++T 토큰 분석기의 rules section을 보여 주고 있다.

5.2 C++T-to-C++ 파서의 설계 및 개발

C++T-to-C++ 파서(이하 C++T 파서) 생성을 위해 기존의 C++ 문법을 확장하고, 이를 Bison을 이용하여 파서 코드를 생성하였다. (그림 7)의 (b)는 C++T 문법을 보여 주고 있다. 또한, (그림 8)는 C++T 번역기를 개발하기 위해 필요한 자료구조를 보여주고 있다.

5.2.1 TAC섹션

TAC을 위한 C++T 문법은 사용하려는 SvM의 이름과 이를 포함한 TMO의 이름을 이용하여 함수를 선언한다. 이러한 방법은 C++ 언어에서 사용하는 방법으로 클래스의 멤버함수를 선언하는 것과 같은 방법이다. 따라서 TMO는 클래스와 같은 의미를 가지며, SvM은 TMO의 멤버함수로 나타낼 수 있다.

5.2.2 ODSS섹션

ODSS를 위한 C++T 문법은 SpM과 SvM이 공유하는 데이터 멤버와 이를 관리하는 멤버함수로 구성된다. 먼저 데이터 멤버의 선언은 SpM과 SvM이 데이터를 공유하는 단위인 segment 단위로 이루어진다. 그리고 ODSS의 멤버함수는 C++와 같은 형태를 갖는다.

5.2.3 SpM섹션

SpM을 위한 C++T 문법은 C++에서 클래스의 멤버함수와 같은 방식으로 구성된다. SpM은 AAC 선언을 위해 'always' 또는 'if-demanded' 키워드를 사용한다 그리고 각 SpM의 바디 부분에는 SpM의 생성자를 위한 코드들을 기술해 준다. 이러한 방법은 C++와 다소 차이가 있는 부분이지만, SpM이 TMO의 멤버함수라는 점에서는 공통점을 갖는다. 그리고 SpM은 AAC절과 SpM을 위한 생성자 코드 부분으로 구성되는데, 이 중 AAC절은 'always' 또는 'if-demanded'로 구성된다

5.2.4 SvM섹션

SvM을 위한 C++T 문법은 SpM과 마찬가지로 C++에서의 멤버함수와 같이 구성된다. SvM 문법의 마지막인 parameter_type_list에는 SvM의 파라메터 선언과 데드라인의 선언이 나타나 있다. 그리고 각 SvM의 바디 부분에는 SvM의 생성자를 위한 코드들을 기술해 준다. 이러한 방법은 C++와 다소 차이가 있는 부분이지만, SvM이 TMO의 멤버함수라는 점에서는 공통점을 갖는다.

5.2.5 TMO 섹션

TMO 클래스를 위한 C++T 문법은 C++T 언어의 새로운 키워드인 'TMO'를 이용하여 나타낸다. 키워드 'TMO'는 C++에서의 'class', 'struct', 그리고 'union'과 같은 역할을 한다. 이러한 TMO 객체는 TMO 모델에서 새롭게 정의된 TMO 실시간 객체를 쉽고 간단하고 명확하게 표현할 수 있도록 구성된다.

6. C++T TMO 프로그래밍

6.1 C++T TMO 프로그래밍

이 장에서는 3장에서 소개한 정수 생성자 TMO와 정수 요청자 TMO에 대한 C++ TMO 프로그래밍을 그대로 C++T TMO 프로그래밍하여 본다.

```
/* 기본적인 자료 구조 */
typedef struct tag_line_t {
  struct tag_line_t *next;

  char   *line;
} line_t;

typedef struct tag_node_t {
  struct tag_node_t *next;
} node_t;

typedef struct tag_param_t {
  struct tag_param_t *next;

  char   *type;
  char   *name;
} param_t;
typedef param_t var_t;
param_t param_temp;
 var_t var_temp;


/* TAC 자료 구조 */
typedef struct tag_TAC_t {
  struct tag_TAC_t *next;

  char   *TMO_name;
  char   *SvM_name;

  char   *return_type;
  param_t *param_list;
  time_t  deadline;
} TAC_t;
TAC_t TAC_temp;


/* ODSS 자료 구조 */
typedef struct tag_ODS_t {
  struct tag_ODS_t *next;

  char   *TMO_name;
  char   *ODS_name;

  var_t  *var_list;
  time_t  MVD;
} ODS_t;
ODS_t ODS_temp;
```

```
/* SpM 자료 구조 */
typedef struct tag_AAC_t {
  struct tag_AAC_t *next;
  char   *SpM_name;
  char   *AAC_name;
  time_t  start, stop, interval;
  time_t  EST, LST, deadline;
} AAC_t;
AAC_t AAC_temp;

typedef struct tag_SpM_t {
  struct tag_SpM_t *next;
  char   *TMO_name;
  char   *SpM_name;
  TAC_t *TAC_list;
  ODSS_t ODSS_list;
  AAC_t  *AAC_list;
  line_t *line_list;
} SpM_t;
SpM_t SpM_temp;

/* SvM 자료 구조 */
typedef struct tag_SvM_t {
  struct tag_SvM_t *next;
  char   *TMO_name;
  char   *SvM_name;
  TAC_t *TAC_list;
  ODSS_t ODSS_list;
  char   *return_type;
  param_t *param_list;
  time_t  deadline;
  line_t *line_list;
} SvM_t;
SvM_t SvM_temp;

/* TMO 자료 구조 */
typedef struct tag_TMO_t {
  struct tag_TMO_t *next;
  char   *TMO_name;
  TAC_t  *TAC_list;
  ODS_t  *ODS_list;
  SpM_t  *SpM_list;
  SvM_t  *SvM_list;
} TMO_t;
TMO_t *TMO=(TMO_t *)NULL;
```

C++T 번역기를 위한 자료 구조

(그림 8) C++T 번역기를 위한 자료 구조
(Fig. 8) Data structure for C++T translator

6.1.1 정수 생성자 TMO의 C++T 코드

(그림 9)의 (a)를 살펴보면, 정수 생성자 TMO는 ODSS, SpM, 그리고 SvM을 각각 하나씩 갖고 있다는 것은 물론, SpM의 시간적인 특성과 SvM에 대한 시간적인 특성까지 쉽게 알아볼 수 있다. 그리고 정수 생성자 TMO를 하나의 실시간 객체로써 간단하게 추상화할 수 있다. 또한 C++에서 객체의 멤버함수를 선언하듯이 정수 생성자 TMO 객체의 SpM과 SvM 및 다른 여러 내부 함수들을 쉽고 간단하게 선언할 수 있음을 알 수 있다.

6.1.2 정수 요청자 TMO의 C++T 코드

(그림 9)의 (b)를 살펴보면, 정수 요청자 TMO는 TAC, SpM, 그리고 SvM을 각각 하나씩 갖고 있다는

```
/* C++T Header */

TMO IntegerGeneratorTMO {
  ODSS:
    segment {
      int K;
    } ODSS1;
    void display(int, int);

  SpM:
    SpM1() {
      always {
        AAC(10hh:10mm:10ss.10msec, 10hh, 250msec)
        start-during(10msec, 35msec) : 200msec;
      };
      closer();
    }

  SvM:
    int SvM1() : 50msec;
};


/* C++T Body */

IntegerGeneratorTMO::display(int x, int y)
{
  gotoxy(x, y);
  printf("%8d", ODSS1.K);
}

IntegerGeneratorTMO::SpM1()
{
  unsigned long t;

  GetCurrentTime(&t);
  ODSS1.K = (int)(t & 0x0000ffffL);
  display(10, 11);
}

int IntegerGeneratorTMO::SvM1() : 50msec
{
  return(ODSS1.K);
}
```

(a) 정수 생성자 TMO에 대한 C++T 프로그램

```
/* C++T Header */

TMO IntegerRequesterTMO {
  TAC:
    int IntegerGeneratorTMO::SvM1() : 50msec;

  SpM:
    SpM1() {
      always {
        AAC(0msec, 1hh, 250msec)
        start-during(10msec, 35msec) : 200msec;
      };
    }

  SvM:
    void SvM1(int) : 50msec;
    {
      /*
       * Here are statements converted to constructor codes.
       */
    }
};


/* C++T Body */

IntegerGeneratorTMO::SpM1()
{
  int i;

  i = IntegerGeneratorTMO::SvM1(BLC) : 40msec;
     // Mode: Blocking Call with 40 msec deadline

  SvM1(10, 13, i , NBC ) : 40msec;
  // Mode: NonBlocking Call with 40 msec deadline
}

void IntegerGeneratorTMO::SvM1(int x, int y, int i) :
50msec
{
  gotoxy(x, y);
  printf("%8d", i);
}
```

(b) 정수 요청자 TMO에 대한 C++T 프로그램

(그림 9) 정수 생성자 TMO 및 정수 요청자 TMO에 대한 C++T TMO 프로그램
(Fig. 9) C++T TMO program for Integer Generator TMO and Integer Requester TMO

것은 물론, SpM의 시간적인 특성과 SvM에 대한 시간적인 특성까지 쉽게 알아볼 수 있고, TAC에 선언되어 있는 호출하려는 함수에 대한 시간적인 정보까지도 쉽게 알아볼 수 있다. 그리고 정수 요청자 TMO를 하나의 실시간 객체로써 간단하게 추상화할 수 있다. 또한 C++에서 객체의 멤버함수를 선언하듯이 정수 요청자 TMO 객체의 SpM과 SvM 및 다른 여러 내부 함수들을 쉽고 간단하게 선언할 수 있음을 알 수 있다.

7. 결론 및 향후 연구 과제

본 논문은 설계 단계에서부터 실시간성을 보장해 줄 수 있는 객체 모델인 TMO 모델을 간단하고 쉽게 명세할 수 있도록 해 주는 C++T 언어를 개발하고, 이러한 C++T 언어로 작성된 코드를 C++ 코드로 번역해 주는 C++T 번역기의 개발에 관한 것이다.

C++ TMO 프로그래밍은 시간적인 표현의 정확성과 단순성을 가지는 TMO 모델을 명세하기에는 그

표현이 복잡하고 어려우므로, 설계 단계에서 TMO 모델이 가지는 실시간성의 보장은 물론 프로그램에 대한 실시간적인 특성의 분석을 어렵게 한다. 그리고 기계적으로 기술해 주어야 하는 많은 초기화 코드 등의 비효율적인 특성 때문에 프로그래머에게 상당한 노력을 요구한다. 그러나 C++ 언어가 보장해 줄 수 있는 객체 지향적인 특성, 기능이 뛰어난 전처리기 및 컴파일러와 같은 도구, 그리고 마지막으로 제공되어지고 있는 TMO 모델과 관련한 라이브러리들이 C++ 언어로 작성되어 있는 점 등을 고려할 때 C++ 언어를 쉽게 버릴 수 없다.

따라서 C++ 언어를 확장하여 TMO 모델의 단순성과 실시간성을 그대로 유지하면서 TMO 모델을 명세 가능하도록 해 주는 새로운 언어인 C++T언어와 이를 C++ 언어로 번역해 줄 수 있는 도구인 C++T 번역기를 개발하였다.

〈표 1〉 C++ TMO 프로그램 코드와 C++T TMO 프로그램 코드의 라인 수 비교

〈Table 1〉 Comparison of C++ TMO program code and C++T TMO program code in terms of the number of lines

	TAC 섹션	ODSS 섹션	SpM 섹션	SvM 섹션	TMO 섹션	전체	비율
C++ 코드	12	44	64	52	17	189	1
C++T 코드	2	4	16	6	18	28	0.15

〈표 1〉은 C++ TMO 프로그램과 C++T TMO 프로그램 코드의 라인 수를 비교하여 본 것을 나타내고 있다. 〈표 1〉에 나타난 바와 같이 C++T TMO 프로그램은 C++ TMO 프로그램에 비해 그 라인 수가 상당히 적다는 것을 알 수 있다. C++T 코드에서 TMO 섹션의 경우 헤더 부분에만 선언되는데, 이는 TAC 섹션, ODSS 섹션, SpM 섹션, SvM 섹션 등을 TMO 클래스 내에 포함하므로 단지 각 섹션의 헤더에 선언되는 코드 수를 합친 것이다. TAC 섹션은 정수 요청자 TMO에 대한 것이고 나머지는 정수 생성자 TMO에 대한 것이다.

본 논문에서 살펴본 바와 같이, C++T TMO 프로그래밍은 TMO 모델의 설계 단계뿐만 아니라 구현 단계에서도 TMO 모델이 가지는 실시간성과 단순성을 유지하며, 실시간적인 특성의 분석을 쉽게 해 준다.

현재 TMO 명세 언어 C++T에 기반한 통합개발환경을 개발하려는 연구가 진행 중에 있다. 이 통합개발환경은 아이콘 기반의 C++T 자동 생성기를 포함하여 TMO 모델을 위한 모든 도구들을 관리해 주기 위한 환경으로써, 도구들간의 주고 받는 내부 중간 코드는 C++T 언어를 기반으로 한다. 그리고 현재 윈도우 NT 상에서 미들웨어 형태로 개발중인 TMO 모델 실행 환경과도 상호작용하게 된다.

참 고 문 헌

[1] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems(WORDS), Oct. '94, Dana Point, pp. 36-45.

[2] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO. k and an Experimental Investigation of Its Potentials", Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC), Nov. 1994, Taipei, pp. 392-402.

[3] Kim, K.H., "A Utopian View of Future Object-Oriented Real-Time Dependable Computer Systems", (Invited paper) Proc. 1st Int'l Workshop on Real-Time Computing Systems and Applications, Seoul, Korea, Dec. 1994, pp. 59-69.

[4] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model-DREAM Kernel-and Implementation Techniques", Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications), Tokyo, Oct. '95, pp. 80-87.

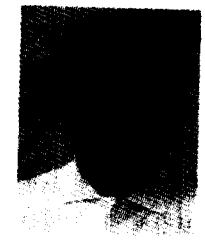
[5] John R. Levine, Tony Mason, and Doug Brown, "Lex & Yacc", O'Reilly & Associates, Inc., Second Edition, Major Revisions, October 1992.

[6] Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986.

[7] Margaret A. Ellis and Bjarne Stroustrup, "The Annotated C++ Reference Manual", Addison-Wesley, 1990.

김 문 희

1979년 2월 서울대학교 전기공학과 학사
1981년 2월 서울대학교 전기공학과 석사
1985년 5월 University of South Florida, MSCS
1991년 5월 University of California, Berkeley, Ph.D
1991년 3월~현재 건국대학교 컴퓨터공학과 부교수
관심분야:실시간 분산처리시스템, 실시간 운영체제, 통신망관리, 소프트웨어공학



박 용 우

1995년 2월 건국대학교 전자계산학과 졸업(공학사)
1997년 8월 건국대학교 전자계산학과 졸업(공학석사)
1997년 9월~현재 건국대학교 컴퓨터·정보통신공학과 박사과정 재학중
관심분야:분산 실시간 시스템, 결함허용, 실시간 운영체제, 실시간 통신망관리, 소프트웨어공학