제 19 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

TMO 프로그래밍을 위한 비주얼 빌더의 설계 및 구현

윤규식, 최재영°, 김문회
건국대학교 컴퓨터공학부
e-mail : dreamery@cse.konkuk.ac.kr

Design and Implementation of a Visual Builder for TMO Programming

Kyusik Yun, Jae-Young Choi°, Moon Hae Kim
School of Computer Engineering and Science, Konkuk University

요 약

TMO 비주얼 빌더는 사용자가 TMO(Time-triggered Message-triggered Object) 모델을 사용하여 실시간 시스템을 개발하는데 도움을 주는 모형화 도구이다. 이 도구는 TMO 의 실시간 요소를 입력해 시각적인 모델링을 하고 거기에 기초한 C++ 소스 코드를 생성시킨다. 이는 초기 설계 작업에서 개발자들에게 편의성을 제공할 수 있고, 이후 안정된 로직을 구현할 수 있도록 한다. 본 논문은 이러한 TMO 비주얼 빌더의 구성 및 기능에 관하여 기술한다.

1. 서론

실시간 시스템은 실제로 설계되기 이전에 시스템이 기능적으로 올바르게 동작해 정확한 값을 산출할 뿐만 아니라 주어진 제약 조건을 위배하지 않음을 보장해야 한다. 하지만, 이러한 실시간 시스템은 분산환경에서 사용되는 통신과 태스크 배정 효과 등이 고려되어야 하므로 그 설계가 더욱 어려워지게 된다.

그러한 시스템을 위해서는 실시간 시스템의 특성을 반영하는 객체 모델이 요구되었다. TMO(Time-triggered Message-triggered Object) 모델은 그러한 요구사항을 충족시킨다. 이 모델은 적시성 서비스 기능(Timely Service Capabilities)을 디자인 단계에서 보장할 뿐만 아니라, 실시간 시스템에서 요구되는 시간에 의한 동기화를 초기 추상화 설계 단계에서부터 실시간 객체로 제공되는 모델이다.

하지만, 이러한 모델을 이용한 실시간 시스템 개발을 할 경우에도 각 실시간 요소들의 입력 및 분석, 그리고 그 행위의 특성 분석을 통한 개발이 필요하게 된다. 따라서 TMO 실시간 요소를 직관적이고, 시각적으로 입력할 수 있는 TMO 비주얼 빌더의 개발이 필요하다. 추가적으로 이러한 TMO 비주얼 빌더는 실시간 미들웨어와 유기적으로 결합되는 한편, 현재 진행중인 프로젝트와 관련하여 확장 가능한 형태로 설계하고 구현되는 것을 보장하는 것이 중요하다.

결국 개발자들은 이러한 툴을 이용해 비주얼한 환경에서 각 요소들을 시각적으로 인식해 입력할 수 있도록 도와주어 신뢰성과 안정성을 보장하는 분산 실시간 처리 응용 프로그램을 개발할 수 있어야 한다.

본 논문은 여러가지 요구 사항을 만족하는 TMO 모델을 기반한 실시간 시스템을 개발하는데 효율적인 모델링 도구인 TMO 비주얼 빌더를 기술한다.

본 논문의 구성은 다음과 같다. 제 2 장은 앞서 제시된 실시간 시스템 환경을 구성하는 TMO 모델을 소개하고 있으며, 제 3 장은 TMO 프로그래밍을 위한 TMO 비주얼 빌더에서의 설계 및 구현을 통해 TMO 실시간 요소를 직관적이고 시각적으로 입력하는 과정과 실시간 시스템 개발의 생산성 향상을 위하여 소스 코드를 자동 생성시키는 과정을 보인다. 마지막으로 제 4 장에서는 TMO 비주얼 빌더의 향후 계획에 대해 설명하고 결론 짓는다.

2. TMO 모델

TMO 모델은 실시간 시스템의 엔지니어가 실시간 시스템 디자인 단계에서부터 실시간성을 제공할 수 있도록 하고 있으며, 실시간 시스템의 추상화에 대한 단순성을 제공하고 있다. 또한, 실시간 시스템의 시간적인 분석을 쉽고 간단하게 해준다.

본 연구는 한국과학재단 목적기초 사업과 대학 IT 연구센터 육성·지원 사업의 지원으로 수행되었음.

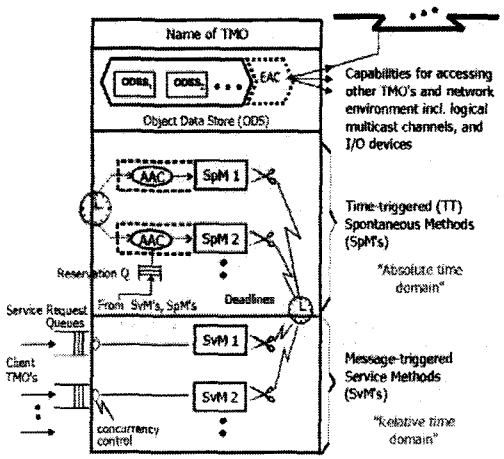

그림 1. TMO 모델의 구조

TMO 모델은 기존의 실시간 객체 모델의 확장으로 이 객체 모델의 기본적인 구조는 그림 1 에 나타나 있다. TMO 모델은 시간적인 분석을 용이하게 해주는 EAC 와 실질적인 구성 요소인 ODSS, SpM, SvM 등의 4 가지 핵심 요소로 구성되어 있다. 다음은 각 요소에 대한 설명이다.

- EAC (Environment Access Capability) : TMO 의 SpM 이나 SvM 에서 다른 TMO 에 있는 SvM 에 대한 호출이 있는 경우, 이를 위한 통신 (communication) 채널을 할당 받아 가지고 있다. 채널을 할당 받기 위해서는 호출하려는 SvM 의 이름과 SvM 이 속한 TMO 의 이름 등이 필요하다. EAC 에 포함되는 정보에는 메시지를 주고 받기 위한 채널의 ID 와 데이터를 주고 받기 위한 저장 장소들이 있다.
- ODSS (Object Data Store Segment) : TMO 의 SpM 이나 SvM 에서 서로 공유하는 데이터를 저장하기 위한 공간으로 세그먼트 단위로 공유되는 ODSS 가 있다. 그리고 이러한 데이터는 MVD (Maximum Validaity Duration)이 지나면 무효한 데이터가 된다.
- SpM (Spontaneous Methods) : TMO 에서 해야 할 작업들은 메소드(Method)로 표현이 되는데, 이중 주기성을 띄거나 시간성을 가지는 메소드 그룹이 SpM 이다. 그리고 이러한 시간적인 특성은 AAC (Autonomous Activation Condition)에 자세히 기술된다.
- SvM (Service Methods) : TMO 의 메소드 중에서 다른 메소드로부터 온 서비스를 수행해 주기위한 메소드 그룹이 SvM 이다. 이러한 SvM 은 고유한 데드라인을 갖는다.

이렇게 TMO 를 구성하는 모든 구성 요소들이 가지는 데드라인(DeadLine)을 디자인 단계에서 정확히 명시해 줌으로써, 그 TMO 에 대한 시간적인 분석을 디자인 단계에서부터 용이하게 해준다.

외부의 클라이언트로부터 온 메시지에 의해 수행되는 SvM 의 실행은 기본적으로 SpM 과 충돌이 없는 경우나, 충돌이 일어난 SpM 의 실행이 끝난 후에만 가능하다. 정확히 말하자면 SpM 과 SvM 사이에는 데이터를 공유하기 위한 ODSS 가 존재하는데, 이를 동시에 액세스하려는 경우에 SpM 이 SvM 보다 더 높은 우선 순위를 가지는 것이다. 그러므로 객체의 수행되는 시간을 디자인 단계에서 고정시킬 수 있고, SpM 의 수행은 SvM 에 의해서 방해 받지 않으며, SpM 의 수행 시 그 시간을 보장 받을 수 있는 것이다.

TMO 의 디자이너는 SvM 의 명세(Specification)에 데드라인을 나타내줌으로써, 클라이언트 객체의 디자이너에게 시간 서비스 능력에 대한 보장을 제공해 줄 수 있다.

3. TMO 프로그래밍을 위한 TMO 비주얼 빌더의 설계 및 구현

시스템에서 필요한 실시간 분산 처리를 위한 실시간 행위 특성 분석과 플랫폼 독립적인 개발환경에서 직관적이며, 효율적인 TMO 실시간 요소를 입력해 소프트웨어를 개발할 수 있어야 한다. TMO 비주얼 빌더는 바로 이러한 요구 사항을 충족시키는 도구이다.

TMO 비주얼 빌더에서 입력해야 하는 TMO 실시간 요소는 다음과 같다.

- SpM 의 AAC 와 보장 데드라인 (Guaranteed Deadline)
- SvM 의 보장 데드라인 (Guaranteed Deadline)
- ODDS 에서 MVD 등의 실시간 데이터에 관련된 실시간 요소
- TMO 간의 통신을 나타내는 SvM 에 대한 호출 관계를 표시하는 EAC

C++ 프로그램을 개발하기 위한 일반 툴은 객체 지향적인 개발 방법을 사용하는 반면, TMO 프로그래밍을 위한 TMO 비주얼 빌더 는 위의 실시간 요소의 입력이 첨가된 개발 방법을 사용해야 한다.

또한, TMO 모델을 기반해 개발되는 실시간 분산처리 시스템의 실시간성 보장은 설계시 정확한 실시간 행위 특성 분석이 필요하고, 그것을 운용할 때는 그 반응이 deterministic 한 실시간 실행 엔진의 지원을 통해서 이루어질 수 있다. 따라서, TMO 비주얼 빌더에서는 이러한 실시간 실행 엔진과 유기적으로 결합되어 이러한 실시간성 및 소스코드 차원의 플랫폼 독립성을 보장해주는 라이브러리인 TMOSL(TMO Support Library)로 코딩이 가능한 형태로 코드를 생성시킬 필요가 있다.

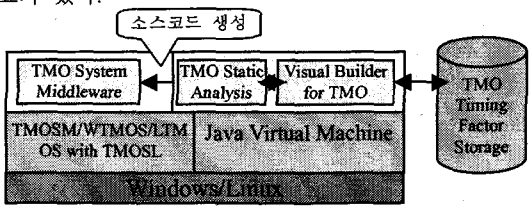

그림 2. TMO 비주얼 빌더 구조의 연구진행

그림 2 는 TMO 비주얼 빌더의 관련된 연구 진행을 보여 주고 있다. 연구 내용에 대한 전반적인 구조이다. 먼저 TMO 실시간성 요소들을 정의하고, 이러한 TMO 의 실시간성 요소들을 입력할 수 있는 기능을 제공해 주는 TMO 프로그래밍 지원 TMO 비주얼 빌더를 개발하고, TMO 비주얼 빌더는 또한, TMO 의 각 실시간 요소들을 바탕으로 정적인 분석을 통한 모니터링 및 변경 작업을 할 수 있다. 현재는 TMO 프로그래밍을 지원하는 TMO 비주얼 빌더를 개발 완료해 TMO 의 실시간성 입력이 가능해 졌다. 이 TMO 비주얼 빌더의 실행환경은 Windows NT 계열상의 TMO 실행 지원 미들웨어인 TMOSM 을 기본적인 플랫폼하고 있지만, 현재 개발되고 있는 Linux 상의 TMO 지원 미들웨어에도 적용이 가능하다. 그러나, 플랫폼에 독립적인 소스를 제공하기 위해서는 다음과 같은 조건을 만족해야 한다.

- 여러 플랫폼의 미들웨어는 TMOSL 를 동일하게 제공해야 한다.
- TMO 비주얼 빌더에서 생성하는 C++ 소스코드에서는 TMOSL 로 제공한다.
- 함수 입력 시 개발자는 TMOSL 만을 사용하여 소스를 생성한다.

실시간 요소 모델링을 위한 형상화는 직관적이고, 빠른 개발 능력을 향상시킨다. 위에서 소개한 TMO 의 4 가지 핵심 실시간 요소를 형상화 시키는 작업 중 그림 3 은 TMO 와 EAC 부분을 실제 TMO 비주얼 빌더에서 형상화 시킨 모습으로 두가지 부분으로 구성되어 있다. 이 TMO 비주얼 빌더의 기본 모델링 원칙은 TMO 관련 문서를 기본으로 두고 설계 및 제작 하였다.

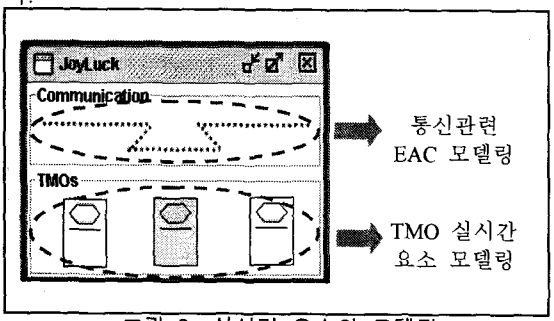

그림 3. 실시간 요소의 모델링

그림 3 에서 상단 부분은 EAC 통신과 관련되어 모델링된 부분이며, 하단 부분은 TMO 들을 나열하여 TMO 의 이름과 main 함수를 포함하는 TMO 와 현재 작업중인 TMO 에 포커스를 두고 작업할 수 있도록 하였다.

TMO 의 개발 과정은 일반 객체 지향 프로그래밍 방법과는 다르다. TMO 는 일단 개발 순서가 정해져 있으며, 구조 또한 정형화 되어 있다. TMO 의 개발 순서를 보면 다음과 같다.

- TMO 를 생성한다.
- TMO 에 속해 있는 각 실시간 요소를 생성한다. (각 실시간 요소는 클래스로 소스가 생성된다)
- 각 실시간 요소의 각종정보(시간 정보, 메소드등)를 입력한다.
- TMO 의 정보(인스턴스 이름, 메소드 등)를 입력한다.

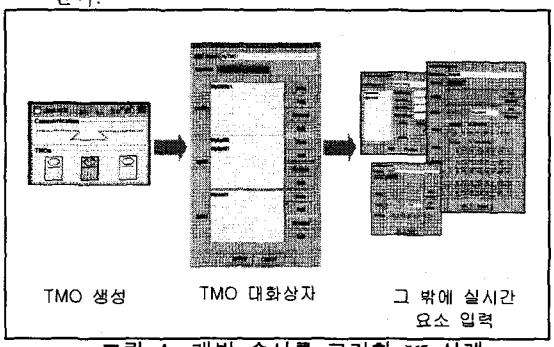

그림 4. 개발 순서를 고려한 UI 설계

그림 4 는 위의 TMO 의 개발 순서를 고려해 디자인하는 과정을 보인다. 그림 3 에서 설계한 패널을 통해 TMO 를 생성하고, 작업할 TMO 를 클릭해 포커스를 맞추어 대화 상자로 작업한다. 그 후에 TMO 대화상자에 포함되는 각종 실시간 요소들을 선택하거나 생성해 값을 입력 하도록 하는데, 세 가지(ODSS, SpM, SvM)의 대화 상자를 통하여 입력 하도록 한다.

마지막으로 이러한 요소들을 입력한 뒤, 코드 생성을 하게 되면 작업중인 프로젝트의 C++코드가 생성되며 그 일부 코드는 아래와 같다.

```
#include "dream.h"
class MyFirstTMOTMOClass{
private:
    ODSSClass1 my_odss1;
    ODSSClass1 my_odss2;
    SvMClass my_svm1;
    SvMClass my_svm2;
    SpMClass1 my_spm1;
    SpMClass1 my_spm2;
    typedef struct{
      int MaxPriority;
      int MinPriority;
      int Velocity;
    } PBRType;
    typedef struct{
      int PID;
      PBRType PBR;
      int Stack_Size;
    }MEPRecType;
public:
    void main(void){
      AIPSupportClass AIP;
      RTOExecManClass RTOMan;
      MEPRecType MEPRec1;
      RTOMan.CreateMEP(5);
      MEPRec1.PBR.MaxPriority = 100;
      MEPRec1.PBR.MinPriority = 0;
      MEPRec1.PBR.Velocity = 10;
      MEPRec1.Stack_Size = 1;
      RTOMan.CreateCustomMEP(&MEPRec1);
      RTO_Class1 RTO;
      RTOMan.BindMethodToCustomMEP( RTO1.SpM1->MID,
MEPRec1.PID);
      if(RTOMan.GetN\umUnboundMEP() > 5)
        RTOMan.BindMethodToMEP(RTO1.SpM2->MID);
      RTOMan.ShareMEP(RTO1.SpM2-> MID.RTO1.SpM3 ->
```

```
MID);
        RTO_Class2 RTO2;
        AIP.PermanentSleep();
    }
}
```

그림 5. 생성된 코드 "MyFirstTMO.cpp"

```
#include "dream.h"
class MyFirstSpM : public BasicSpMClass{
private:
    ODSSClass1 *my_odss1;
    ODSSClass1 *my_odss2;
    void SpMBody(void){
        unsigned long Timestamp;
        TimeOfDay Reply_Deadline;
        NonBlockingSRQ(RAC->DFCID_RTO2_SvM3,    &RAC-
>Par_RTO2_SvM3,
Sizeof( RTOClass1::RTOAccessCapabilityClass::ParType_RTO2
_SvM3), &Timestamp);
        Status = NonBlockingGetResultOfNonBlockingSRQ
          (&RAC->Par_RTO2_SvM3, Timestamp);
        if(Status != SUCCESS)
          BlockingGetResultOfNonBlockingSRQ(          &RAC-
>Par_RTO2_SvM3, Timestamp, Reply_Deadline);
        ReportCompletion();
    }
public:
    MyFirstSpM(ODSSClass1      *my_odss1,      ODSSClass1
*my_odss2 ){
        this.my_odss1 = my_odss1;
        this.my_odss2 = my_odss2;
    }
};
```

그림 6. 생성된 코드 "MyFirstSpM.cpp"

```
#include "dream.h"
class MyFirstSvM:public BasicSvMClass{
private:
    ODSSClass1 *my_odss1;
    ODSSClass1 *my_odss2;
    typedef struct{
        int object_id;
        double speed;
    }SvMParameterType;
    SvMParameterType SvMParameter;
    void SvMBody(void){
        int Client_DFCID_RR;
        unsigned long Timestamp;
        TimeOfDay SRQ_Deadline;
        ReceiveSRQ(&Client_DFCID_RR,          &SvMParameter,
&Timestamp);
        BlockingSRQ(RAC->DFCID_RTO2_SVM3,          &RAC-
>Par_RTO2_SvM3,
sizeof( RTOClass1::RTOAccessCapabilityClass::ParType_RTO2_
SvM3), SRQ_Deadline);
        ReplySRQ(Client_DFCID_RR, &&SvMParameter,
          sizeof(SvMParameterType), Timestamp);
        ReportCompletion();
    }
public:
    MyFirstSvM(ODSSClass1      *my_odss1,      ODSSClass1
*my_odss2 ){
        this.my_odss1 = my_odss1;
        this.my_odss2 = my_odss2;
}};
```

그림 7. 생성된 코드 "MyFirstSvM.cpp"

4. 결론 및 향후 연구과제

체계적이고 일관적이며 통합적인 실시간 요소에 대한 정의, 입력, 구현에 있어서 TMO 모델이 가지는 장점에 근거해 미들웨어 지원 환경 및 실행 환경이 개발 되었으며, 이를 토대로 모델링 도구인 TMO 비주얼 빌더를 개발하였다. TMO 프로그래밍을 위한 TMO 비주얼 빌더를 통해서 TMO 프로그래머는 시뮬레이션을 직접 모델링 하는 과정에 있어서 손쉬운 모델링이 가능하게 되었고, 구현하는 과정에 있어서도 각각의 TMO 에 대해서 TMO 라이브러리에 기반한 소스코드 생성 및 시간 제약 조건에 해당되는 실시간 요소들의 입력을 TMO 비주얼 빌더를 통해서 제공 받을 수 있게 되었다.

TMO 비주얼 빌더에 의해 자동 생성된 TMO 소스코드를 가지고, 효율적인 프로그래밍이 가능하게 되었고, 프로그래밍 작업에 있어서의 시간 손실을 줄여 디자인 단계에 좀더 충실한 시스템 설계 및 모델링을 할 수 있게 되었다.

추가적으로 각 TMO 컴포넌트의 시간 제약조건을 명확히 검증하는 기능의 첨가가 필요하며, 이는 각각의 TMO 컴포넌트에 대해 시간 상태 분석기 등이 정적 및 동적인 상태에서 해당 TMO 에 최적화된 시간 제약 조건을 계산해 분석 함으로서 TMO 객체가 SpM 및 SvM 등의 실시간 요소들 간의 효율적인 사용이 가능하도록 하는 것은 향후 과제로 남긴다.

참고문헌

[1] Kopetz, H. and Grunsteidl, G., "TTP-A: Time Triggered Protocol for Fault-Tolerant Real-Time Systems", Proc. IEEE CS FTCS-23, Toulouse, June 1993, pp. 524-533

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

[3] Kim, K.H., Ishida, M., and Liu, J., "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", Proc. ISORC '99 (IEEE CS 2nd Int'l Symp. on Object-oriented Real-time distributed Computing), May 1999, pp.54-63

[4] Puschner, P., "A Tool for High-Level Language Analysis of Worst-Case Execution Times", Proc. of the 10th Euromicro Workshop on Real-Time Systems, Berlin, Germany, 1998

[5] 박용우, 김문회, 김정국, "TMO 모델 기반 실시간 시뮬레이션", 정보처리학회지, 제 5 권, 제 4 호, pp.67-74, 한국정보처리학회, 1998.7

[6] J.G., Kim, M.H., Kim, B.J. Min, D.B., Im, "A Soft Real-Time TMO Platform - WTMOS - and Implementation Techniques", Proc. 1st IEEE International Symposium on Object-oriented Real-time Distributed Computing, Kyoto, Japan, April 20-22, 1997.

[7] 자바에서 한글 입력을 위한 환경 설정 방법 http://kr.geocities.com/baram4x/