TMO를 기반으로한 분산환경에서의 실시간 항공기 착륙 전산 모사 시스템의 구현

Implementation of a TMO Based Real-Time Airplane Landing Simulator on a Distributed Computing Environment

Implementation of a TMO Based Real-Time Airplane Landing Simulator on a Distributed Computing Environment

by

Min-Gu Lee

Division of Electronic and Computer Engineering

(parallel processing program)

Pohang University of Science and Technology

A thesis submitted to the faculty of Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Department of Electronic and Computer Engineering (parallel processing program).

Pohang, Korea

12. 28. 2001

Approved by

_____

Sunggu Lee

Major Advisor

TMO를 기반으로한 분산환경에서의 실시간 항공기 착륙 전산 모사 시스템의 구현

이 민 구

위 논문은 포항공과대학교 대학원 석사 학위 논문으로 학위 논문심사위원회를 통과하였음을 인정합니다.

2001년 12월 28일

심사위원장 이 승 구 (인)

위원 홍 성 제 (인)

위원 김 종 (인)

MECE 20002417 이 민 구, Min-Gu Lee, Implementation of a TMO Based Real-Time Airplane Landing Simulator on a Distributed Computing Environment, TMO를 기반으로한 분산환경에서의 실시간 항공기 착륙 전산 모사 시스템의 구현, 전자컴퓨터공학부 (병렬처리전공), 2001, 101P, Advisor : Sunggu Lee. Text in Korean.

ABSTRACT

In real-time simulation, the system being simulated should display the same timing behavior as the target system. The simulation accuracy is increased as the simulation time unit is decreased. Although there are several models for such a system, the TMO model is particularly appropriate due to its natural support for real-time distributed object oriented programming. This thesis discusses the results of the implementation of a real-time airplane-landing simulator on a distributed computing environment using the TMO model.

차례

표 차례

그림 차례

1. 서론

전산 모사(Computer Simulation)는 물리적인 한계, 시간적 제약, 실제 시스템의 구현의 어려움, 편리성 등의 여러 이유로 인해서 전산기(Computer)를 이용하여 실제 상황/시스템을 모델링(Modeling)하고 그 모델에 따라서 어떠한 작업을 수행하여 실제 시스템에서의 상황을 알아보고자 하는 것이 목적이다. 실시간 컴퓨팅(Real-Time Computing)이란 계산을 수행할 때 그 결과뿐만 아니라 결과가 도출되기까지의 시간에 제약을 주어 수행하는 것을 의미한다. 실시간 전산 모사는 위의 두 개념을 더한 것 즉, 실시간으로 동작하는 전산 모사를 말한다.

1.1. 전산 모사의 분류

실시간 전산모사는 스케일 된 전산 모사 (scaled real-time simulation)와 스케일 되지 않은 전산 모사(non-scaled real-time simulation)로 나뉜다. 스케일 된 실시간 전산 모사는 실시간으로 전산 모사를 수행하나 빠른 결과를 보기 위해서 혹은 보다 정확한 결과를 위해서 그 시각 단위(time unit)가 스케일 된 경우를 이야기한다. 전자는 시각 단위가 줄어야 할 것이고, 후자는 늘어나는 경우가 될 것이다. 화학반응의 합성 과정에 대한 전산 모사가 스케일 된 전산 모사의 좋은 예가 된다. 화학 반응을 실제 화학 반응이 일어나는 것과 똑같은 시각 단위로 전산 모사를 수행해야 하는 이유가 없을지라도, 화학약품이 합성되는 과정을 자세히 보여주기 위해서는 축척에

의해 비례되어진 시각 단위로 수행한 후에 결과를 살펴보는 것이 좋을 것이다. 특히 어떠한 화학 반응은 반응 시간이 몇 시간에서 심지어는 수 일간 수행되는 경우도 있으므로, 이러한 경우에는 빨리 결과를 얻어내는 것이 매우 필요하다.

스케일 되지 않은 실시간 전산 모사는 목적 시스템과 같은 시간적 지연 현상을 나타내는 속도로 전산 모사가 수행되어야 하는 것이다. 이러한 실시간 전산모사는 실시간 교통 제어 시스템이나, 항공기 조정사 모의 훈련 시스템처럼 사람이 어떠한 외부 사건에 대해서 반응을 해야 하는 경우에 매우 유용하다. 항공기 조정 전산 모사 시스템을 만들 때 개발된 시스템은 반드시 모든 상황을 실시간으로 전산모사를 수행해서 훈련에 참가해야 하는 항공기 조정사는 실제 항공기를 조정하는 상황과 동일한 시간 지연 특성을 경험하도록 해야 하는 것이다.

스케일 되지 않은 실시간 전산모사를 수행하는데 있어서 계산 속도를 가능한 한 증가시키는 것뿐만 아니라 실시간 작업(real-time task)을 예측 가능한 지연 시간 내에 처리할 수 있어야 하며, 불규칙적이지 않아야 하는 것 또한 매우 중요하다. 즉, 실시간 작업이 수행되는 시각이 시스템의 부하(load)나 기타 다른 이유로 인해서 변화되면 안되는 것이다.

1.2. 전산 모사를 지원하기 위한 도구들

지금까지 개발/시판 되어 지고 있는 상용 전산모사 도구들은 실시간 전산모사 시스템을 개발하기에 충분한 기능들을 제공하고 있지 않다. 몇몇의 도구들은 분산 환경의 지원을 하지 않기도 한다. 실시간 전산 모사의 정확도

를 증가시키기 위해서는 계산량의 증가를 감수하더라도 보다 정확한 모델링 방법을 사용하는 것이 필요하고, 전산 모사의 시간 주기를 가능한한 최소화 시키는 것이 필요하다. 이러한 조건을 만족하기 위해서는 전산기의 계산 용량의 증가가 필요하게 된다. 그러나 전산기의 계산 용량의 한계가 있으므로 이를 극복하기 위해서는 분산 환경의 지원이 필요하게 된다. 또한 프로그램을 작성하는데 있어서 객체 지향으로 작성하는 것은 프로그램을 보다 간단히 작성하기 위해서 반드시 필요하게 된다.

이러한 요건들을 만족하는 도구들로 [3, 4, 5]가 있다. 이러한 도구들 가운데 우리는 TMO (Time-triggered Message-triggered Object) 모델[3, 5, 6]을 사용하였다. 그 이유는 TMO 모델이 객체 지향형 실시간 전산모사를 자연스럽게 지원하기 때문이다. 이 논문에서는 TMO를 살펴보고 실시간 항공기 전산 모사 시스템을 구현함으로서 TMO를 이용한 실시간 전산 모사의 구현에 대해 논의할 것이다.

1.3. 관련 연구

기존의 TMO를 이용하여 구현된 실시간 전산모사들은 스케일 되어진 전산 모사이거나, 사용자의 입력과 상관없이 정해진 각본대로 수행되는 전산 모사였다. 해상의 함선을 지키는 전략에 대한 전산 모사 시스템인 CAMIN[2]이나, 고속도로의 교통을 전산 모사하는 DOFS[1]등이 있다. 이 시스템들은 모두 TMO를 이용하여 분산 환경에서 동작하도록 구현 되어 있다.

CAMIN에서는 해상의 함선을 외부의 공격으로부터 방어하는 모의 해상 전쟁 전산 모사 시스템에 대하여 연구하였다. 이 전산 모사 시스템은 전산

모사 하고자 하는 공간 외부에서부터 적의 공격이 들어오는 것을 탐지하고 그에 대한 적절한 방어를 수행하는 것이 목적이다. 이 전산 모사 시스템은 방어 시스템의 각본이 실제 외부 공격에 대한 방어가 성공적으로 수행될 수 있는지를 전산 모사 하는 응용 프로그램이고, 스케일 되어진 전산 모사 시스템이다. 그러므로 실제 외부에서의 공격에 대해서 동일한 시각단위로 동작 하는 것이 아니라, 보다 빠른 결과를 얻기 위해서 시각 단위가 스케일되어진 전산모사 시스템이다. 이 시스템은 전산 모사를 수행하고자 하는 대상 공간과 공격을 하는 외부 공간, 그리고 실제 방어 시스템을 제어하는 시스템 등으로 대상을 top-down 설계 방법론을 이용해서 TMO 모델을 적용하여 모델링 하고, 이를 구현하였다. 이 연구는 TMO 모델이 스케일 된 실시간 전산 모사 시스템에 적합함을 보여주었다.

DOFS에서는 고속도로의 교통 제어에 관련된 전산 모사를 수행하였다. 이 연구에서는 고속도로의 진입 통제가 교통 흐름에 어떠한 영향을 미치는지에 대한 전산 모사 시스템이었다. 이 전산 모사 시스템도 역시 TMO모델을 이용하여 구현하였으며 고속도로를 지역별로 분산시켜 분산 환경에서 수행될 수 있도록 구현되었다. 각각의 노드는 인접한 노드와 통신을 통해서 정보를 주고 받으면서 실시간으로 고속 도로상의 교통 흐름을 전산 모사 하였다. 이 연구에서는 고속 도로상의 각각의 자동차들이 개별적으로 모델링 되어서 전산 모사가 수행되었으므로 그 계산량이 매우 크다. 뿐만 아니라 자동차는 수초 내에 그 상태에 큰 변화가 발생할 수도 있으므로 시각 단위가 매우 작아야 한다. 그러나 앞에서 언급한대로 계산량이 매우 크므로 하나의 노드에서 전산 모사를 수행하면 그 전산 모사의 시각 단위가 요구치보다 커지게 되므로 분산 환경에서의 수행이 필요하게 되고, 실제 분산 환경에서 구현함으로써 TMO모델이 분산 환경을 효율적으로 지원함으로써 부하가 크게 걸리는 응용 프로그램에 적합함을 보였다. 그러나 이 연구에서는 사용자와의 연계 없이 이

미 정해져 있는 각본대로 전산 모사를 수행하는 응용 프로그램이었다. 이는 스케일 되지 않은 전산 모사의 사용자와의 연계가 필요한 응용 프로그램에는 어떻게 적용 되는지 알 수 없다.

1.4. 연구 목표

앞에서 언급한대로 스케일 되지 않은 전산 모사 시스템은 사용자와의 연계가 필요한 전산 모사 시스템에서 가장 중요하다. 일례로 항공기 조정사를 훈련하는 시스템을 구현하는 경우 조정사는 실제 시스템과 동일한 시간적 지연 현상을 느낄 수 있도록 해야 한다. 뿐만 아니라 조정사는 전산 모사가 연속적으로 진행되는 것처럼 느끼기 위해서는 그 주기가 수십 밀리 초 정도로 구현되어야 한다. 본 논문에서는 기존의 연구들에서 다루지 못한 사용자와의 연계가 필요한 응용 프로그램을 작성하고, 그 성능을 평가함으로써 TMO모델이 분산 환경에서 사용자와의 연계가 있는 실시간 전산 모사 시스템의 구현에 얼마나 적합한가를 평가해 보고자 한다.

본 논문은 우선 2절에서 TMO 모델에 대한 간략한 소개를 한 후에, 3절에서 TMO 모델이 목적 응용프로그램에 적합한 성능을 보이는지 평가를 할 것이다. 그리고 4절에서는 목적 응용프로그램을 TMO 모델을 이용하여 모델링을 수행하고, 5절에서는 수행된 모델을가지고 실제 응용 프로그램을 구현한다. 6절에서는 구현된 응용 프로그램의 성능을 살펴보고, 목적 응용 프로그램의 요구조건을 만족시키는지 살펴 본 후에, 마지막으로 7절에서 결론을 내릴 것이다.

2. TMO 모델의 개괄

2.1. 개괄

TMO model은 미국의 UCI(University of California, Irvine : 얼바인 소재 캘리포니아 주립 대학)의 김광회 (K. H. Kim) 교수가 개발한 것으로서 1990년대 초반에 개발된 RTO.k [7, 8]를 기반으로 보다 확장한 것이다. 이는 실시간 시스템을 효율적으로 구축하기 위한 구체적인 프로그래밍 방법론이다. 이번 절에서는 본 논문을 이해하기 위해서 필요한 TMO 모델의 개괄에 대해서만 살펴 보도록 한다. 보다 자세히 알고 싶은 사람들은 [7, 8]의 논문을 참조하기 바란다. TMO 모델은 [1]에 나와있는 것처럼 다음에 설명하는 기능들을 가지고 있다.

1) 분산 처리를 지원하는 객체를 사용: TMO 모델은 분산 환경을 지원한다. TMO 객체는 여러 노드에 나뉘어져서 원격 함수 호출 (Remote Method Call)을 이용하여 TMO간의 정보 교환을 하고, 상호 작용을 하게 된다. 이때 클라이언트 TMO가 서버 TMO에게 서비스 요청을 하는 경우에 비봉쇄(Non-blocking) 요청을 통해서 동시성 (concurrency)를 높일 수 있다.

2) 두개의 메소드의 분리 : TMO 모델에서는 SpM (Spontaneous method) 과 SvM (Service method)으로 명확히 구분되는 두개의 메소드를 가지고 있다. SpM은 내부 실시간 클럭에 의해서 촉발되며, SvM은 외부에서 전달되어 온 메시지에 의해 촉발된다. 그래서 SpM은 시간에 의해 촉발되는 메소드 (time-triggered method)라고 불리우고, SvM은

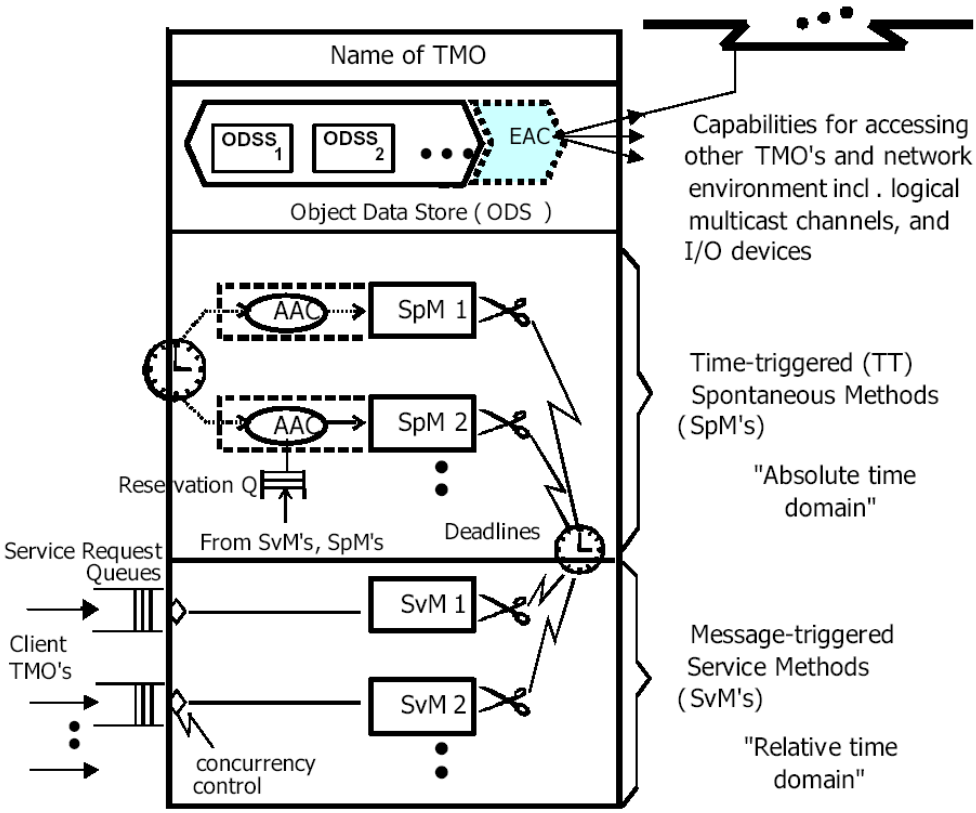


그림 1 TMO의 구조 ([1]에서 발췌).

메시지에 의해 촉발되는 메소드(message-triggered method)라고 불리운다.

3) Basic concurrency constraints (BCC) : BCC는 SpM과 SvM의 충돌 가능성을 막기 위한 방법이다. 만약에 SpM과 SvM이 같은 ODS(Object Data Store)의 변수에 접근한다면 SvM은 그 충돌이 없어질 때까지 그 수행이 연기되어진다. 즉 SvM은 SpM과의 충돌이 없는 경우에만 수행될 수 있도록 하는 것이다.

4) 완료시각의 보장 : TMO 모델은 AAC(Autonomous Activation Condition)에 의해서 정의되는 시간창(time window)을 지원함으로서 수행 완료시간을 보장하도록 한다.

2.2. TMO 객체의 구조

TMO의 구조는 그림1 과 같이 ODS (Object Data Store), SpM, SvM, EAC (Environment Access Capability), AAC (Autonomous Activation Condition)으로 이루어져 있다. 각각의 역할은 [1]에서 언급 되어 있으며, 아래에 요약, 설명한다.

ODS : 사물 혹은 현상을 객체로 모델링 할 때 필요한 상태, 속성 등의 자료의 저장소.

SpM : 이미 앞에서 언급한대로 내부 실시간 클럭에 맞추어서 촉발되는 메소드로 실시간으로 처리해야 하는 일을 담당하는 메소드

SvM : 외부 메시지에 의해서 촉발되는 메소드로 다른 TMO에서

들어오는 메시지를 처리하는 메소드.

EAC : TMO의 ODS를 다른 TMO간의 접근 방법.

AAC : SpM이 실행되어야 하는 시간에 대한 조건으로 시간창(time window)을 설정.

2.3. TMO의 지원

TMO 모델을 지원하기 위해서 UCI의 DREAM 연구실에서는 TMOSM(TMO support middleware)과 TMOSL(TMO support library)[9]을 개발하였다. TMOSM은 TMO 모델을 이용하여 작성된 응용프로그램들을 COTS 운영체제에서 수행시키기 위한 미들웨어 모델로서 현재는 Microsoft Windows NT 환경에서 수행되도록 구현되어 있으며 TMOSM/NT라고 불리운다. TMOSL은 TMO 모델로 응용프로그램을 작성하기 위한 사용자 수준의 API이다. TMOSL은 몇 개의 C++ 클래스들로 구성되어 있다.

TMOSM/NT는 TMO모델을 Microsoft Windows NT환경에서 수행시킬 수 있도록 개발된 미들웨어이다. 이 TMOSM/NT의 내부 구조는 그림 2에서 볼 수 있다. [9]에서 언급한대로 TMOSM/NT에는 미들웨어 쓰레드와 응용 쓰레드의 두 묶음으로 나뉘어진다. 이 쓰레드들은 그림 2에서 볼 수 있듯이 WTST에 의해서 선택되어진다. 미들웨어 쓰레드는 4개의 쓰레드로 구성되며 각각은 아래에 설명된다.

WTST (Watchdog Timer & Scheduler Thread) : 이 쓰레드는 일정주기 (Windows NT의 watchdog timer에 설정해 놓은 주기 - 현

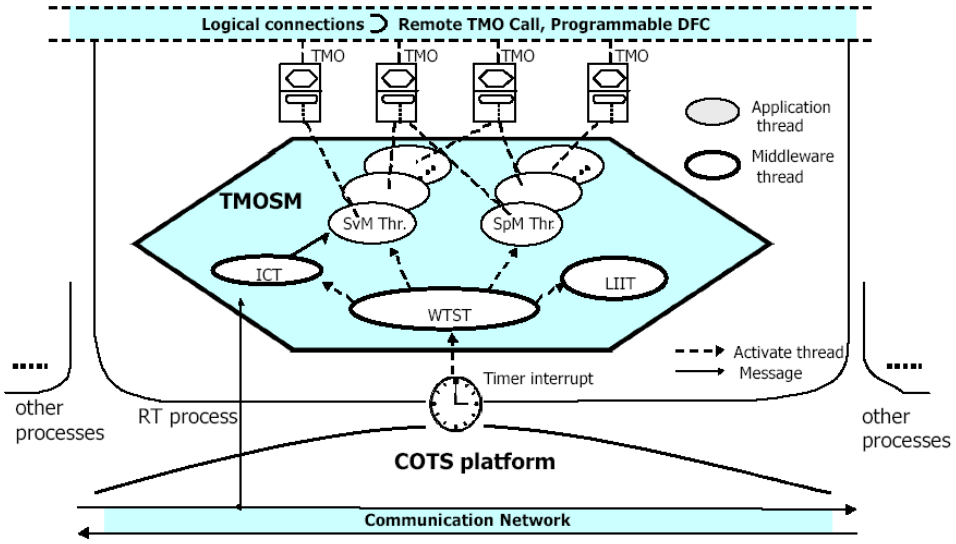


그림 2 TMOSM의 내부 구조 ([9]에서 발췌).

재의 TMOSM에서는 3 ms)마다 실행되면서 TMOSM에 포함되는 다른 모든 쓰레드의 실행을 관장한다.

LIIT (Local I/O Interface Thread) : 이 쓰레드는 디스크 I/O같은 지역 I/O를 관장하는 쓰레드이다.

ICT (Incoming Communication Thread) : 이 쓰레드는 네트워크를 통해서 들어오는 통신 메시지를 그 목적지 쓰레드로 보내주는 역할을 담당하는 쓰레드이다.

VMST (Virtual Main System Thread) : 이 쓰레드는 실제 응용 프로그램에 할당되는 쓰레드이다. 이 쓰레드에 CPU 시간이 할당되면

응용 쓰레드 스케쥴러가 호출되어 실제 수행할 SvM이나 SpM을 선택한 후에 그 쓰레드가 수행된다.

3. TMO 프로그래밍 환경의 성능 평가

실제로 TMO 모델을 이용해서 항공기 착륙 전산모사 시스템을 만들기 전에 우리는 현재의 TMO 구현이 우리의 필요를 만족할 수 있는지를 알아 보기 위하여 시험해 보았다. 모든 실험은 128 Mbytes의 주 기억장치를 갖고, 100Mbps 의 고속 이더넷(fast ethernet)에 연결되어 있는 Intel Pentium III 733Mhz PC 클러스터에서 수행되었다. 클러스터에 사용되는 OS는 Microsoft Windows 2000 professional이고, TMOSL과 TMOSM/NT가 설치되어있다.

3.1. SpM이 실제로 수행된 시각과 예상 수행 시각의 차이

TMO모델에서 SpM은 실제 수행된 시각과 예상되는 수행 시각의 차이를 최소화해서 수행되어야 한다. 표 1은 한 노드에서 하나의 TMO에 있는 여러개의 SpM이 수행될때 그 예상 수행 시각과 실제 수행된 시각의 차이를 최소값, 최대값, 평균값 그리고 표준편차를 보여준다. 이 결과들은 한 노드에서 10,000번을 수행시킨 결과값이다.

표 1의 결과를 살펴보면 평균적인 실제 수행시각과 예상 수행 시각의 차이가 약 4.6 ms임을 볼 수 있다. 그리고 이 평균값은 SpM의 주기와는 상관이 없음을 볼 수 있다. 표 1은 또한 SpM의 수가 증가함에 따라서 아주 약간 평균값이 증가함을 볼 수 있다. 최소 값은 약 40 us이고, 최대 값은 약 9.5 ms이다. 만약에 SpM의 주기가 매우 작다고, SpM의 수가 증가하게 되면

표 2 SpM의 수에 따른 한 TMO의 실제 수행시각과 예상 수행 시각의 차이. (*N/A : 실험 불가, 단위 : microseconds).

주기	SpM의 수	최소값	최대값	평균값	표준편차
16000	1	36	9061	4546.329	2604.701
	2	41	9103	4569.989	2604.13
	3	*N/A	*N/A	*N/A	*N/A
	4	*N/A	*N/A	*N/A	*N/A
30000	1	39	9062	4545.992	2604.692
	2	45	9102	4572.700	2602.834
	3	45	9146	4593.956	2601.195
	4	53	9190	4616.444	2599.624
50000	1	41	9066	4551.841	2604.294
	2	40	9174	4572.965	2602.477
	3	46	9154	4596.086	2600.252
	4	51	9191	4617.581	2598.559
75000	1	36	9071	4548.639	2604.454
	2	45	9109	4576.087	2602.609
	3	48	9161	4597.516	2601.100
	4	53	9213	4618.486	2599.403
100000	1	43	9484	4550.820	2604.705
	2	50	9120	4576.519	2602.573
	3	49	9165	4597.524	2600.825
	4	53	9200	4619.558	2599.003

TMOSM은 SpM을 제대로 수행시킬 수 없다. 이 제한점의 이유는 현재의 TMOSM의 문제인 듯 하다. 이 결과는 현재의 TMO 구현이 매우 낮은 전산 모사 시각 단위(약 10 ms 이하의)를 가지는 응용프로그램에서는 적당하지 않음을 볼 수 있다. 그러나 최대값이 약 9.5 ms의 범위 내에 있으므로, 수십 밀리 초 단위의 전산 모사 시각 단위를 갖는 응용프로그램에는 적당함을 알 수 있다. 그러므로 본 논문에서 구현하고자 하는 응용프로그램에서는 수십 밀리 초 단위의 주기를 갖기 때문에 충분한 성능을 보인다고 할 수 있다.

표 2는 하나의 SpM을 갖는 여러개의 TMO를 하나의 노드에서 수행시켰을 때의 결과를 보여준다. 이 결과를 살펴보면 역시 TMO의 개수와 실제 수행시각과 예상 수행시각의 차이는 관계가 없음을 볼 수 있다. 최대값은 표 1에서의 결과처럼 TMO의 수가 증가함에 따라서 조금씩 증가하나 그 양이 고려의 대상이 되지 않을 정도로 매우 작은 것을 볼 수 있다.

3.2. 여러 노드간의 수행시각의 차이

TMO 모델은 분산 컴퓨팅 환경을 지원한다. 즉, TMO 객체들을 여러 노드에 분산 시키고 수행시킬 수 있다는 것이다. 그러므로 각각의 노드에서의 수행 시각의 차이는 매우 중요한 문제가 된다.

그림 3에서는 동일한 TMO를 두 노드에 나누어 놓았을 때의 수행시각의 차이를 보여준다. 가로축은 시간 혹은 단계를 나타내고 세로축은 두 노드간의 수행시각의 차이를 나타낸다. 그림 3은 16 ms의 주기로 SpM을 10,000번 수행시켰을 때의 결과이다. 실험을 수행하는 동안 지역 네트워크에 다른 통신 부하는 없었다. 그림에 보면 정확히 8번의 9 ms에 가까운 차이를

표 3 한 노드에서 TMO의 수에 따른 실제 수행시각과 예상 수행시각의 차이. (*N/A 실험 불가, 단위 : microseconds).

주기	TMO의 수	최소값	최대값	평균값	표준편차
16000	1	36	9061	4546.329	2604.701
	2	41	10623	4569.707	2605.151
	3	*N/A	*N/A	*N/A	*N/A
	4	*N/A	*N/A	*N/A	*N/A
30000	1	39	9062	4545.992	2604.692
	2	43	9104	4571.959	2602.785
	3	47	9177	4591.875	2601.127
	4	53	9223	4616.971	2599.350
50000	1	41	9066	4551.841	2604.294
	2	40	10484	4573.991	2602.420
	3	50	10282	4597.514	2600.338
	4	53	9318	4616.902	2598.169
75000	1	36	9071	4548.639	2604.454
	2	42	9116	4572.442	2602.431
	3	46	9155	4595.169	2600.978
	4	51	10109	4617.263	2599.534
100000	1	43	9484	4550.820	2604.705
	2	47	10422	4574.093	2602.821
	3	48	9172	4597.599	2600.521
	4	58	9199	4622.734	2598.668

보이는 것을 볼 수 있다. 그 외에는 매우 작은 차이를 보이는 것을 볼 수 있다. 이러한 큰 차이는 TMOSM/NT가 Windows NT를 완전히 제어할 수 없기 때문에 발생하는 문제이다. TMOSM/NT는 Windows NT의 커널을 수정하지 않았기 때문에 실시간 운영체제가 아니므로 그림 3에서의 결과처럼 간혹 큰 차이를 보이기도 한다. 있다. 그러나 앞으로 표 3을 살펴보면 알 수 있듯이 그러한 경우는 매우 적고 평균값이 매우 작고, 또한 실제 실시간으로 동작하는 주기가 현재의 TMOSM/NT의 구현에서는 16 ms를 넘어야하므로, 9 ms의 차이는 수용할 수 있는 정도이다. 또한 본 논문에서 구현하고자 하는 목적 응용 프로그램에서도 역시 충분히 수용할 수 있는 오차범위 내에 있다.

표 4 여러 노드에 분산된 SpM들의 수행 시각의 차이. (단위 : usec).

노드 수	최소값	최대값	평균값	표준편차
2	0	9019	12.117	254.647
3	0	9020	12.039	190.311
4	0	9034	9.560	186.340

표 3은 여러 노드에 분산되어 있는 TMO의 SpM들의 수행 시각의 차이를 보여준다. 모든 값들은 기준 노드와 각각의 다른 노드들의 차이 중에서 가장 큰 값을 취한 것이다. 표를 살펴보면 노드수와 각각의 노드간의 수행시각의 차이는 상관관계가 없음을 알 수 있다. 평균값은 약 12 마이크로 초이고, 최소 값은 0이다. 0이라는 것은 모든 노드에서 수행된 시각의 차이가 없다는 것으로 완벽한 시스템을 의미한다. 최대값은 약 9 ms 이고, 이 값은

TMOSM/NT가 Windows NT를 완전히 제어하지 못하는 것에 기인한다. 표준 편차가 300 마이크로 초 이하로 매우 작으므로 노드간의 수행 시각의 차이는 매우 작음을 알 수 있다. 최대값이 약 9 ms 정도이므로 수십 밀리 초 이상의 주기를 갖는 응용프로그램에는 적합하다는 것을 볼 수 있다.

3절의 결과들을 살펴보면, 현재의 TMO 모델과 그 구현이 분산 환경에서 수십 밀리 초 단위의 응용프로그램을 분산 환경에 적용시키기 적당하다는 것을 알 수 있다. 본 논문의 목적 응용 프로그램이 역시 수십 밀리 초 단위를 갖도록 구현되므로 그 성능이 충분함을 볼 수 있다.

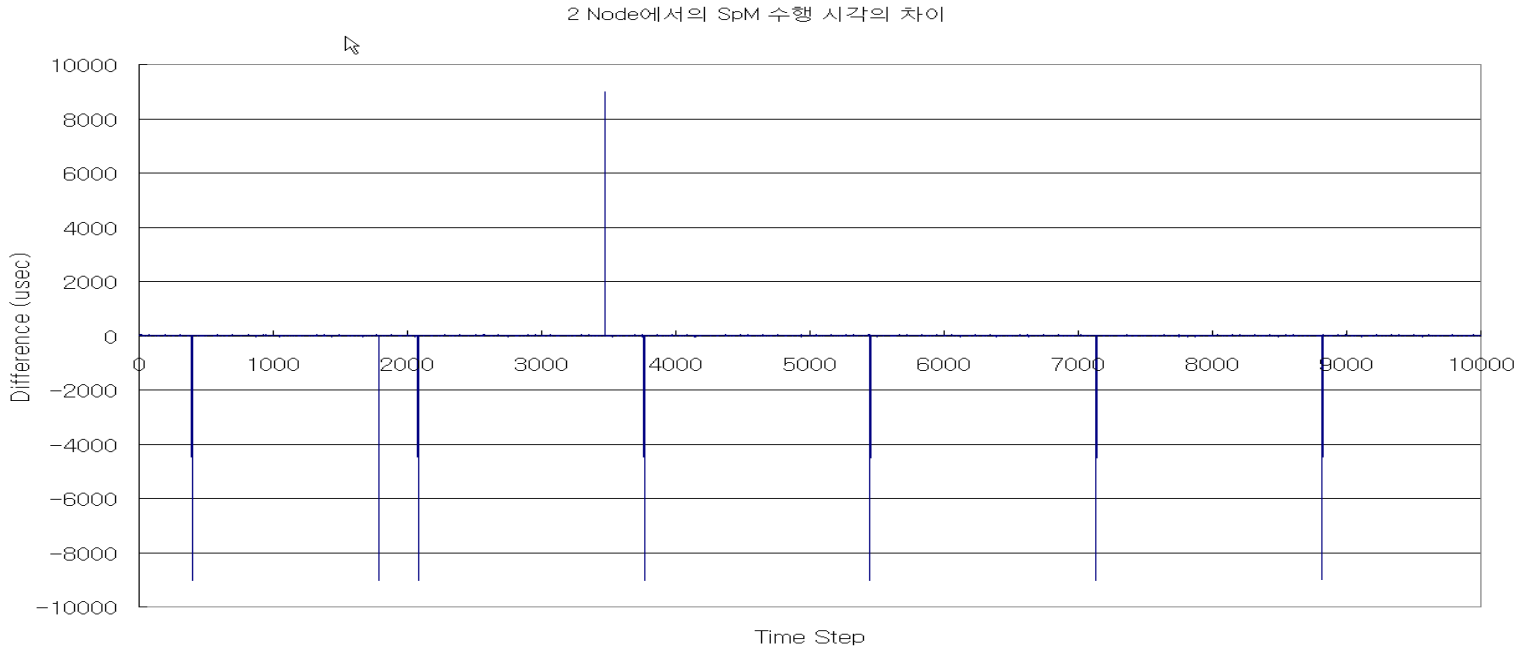


그림 3 두 노드간의 SpM 수행 시각의 차이. (단위 : microseconds).

4. TMO 모델의 적용

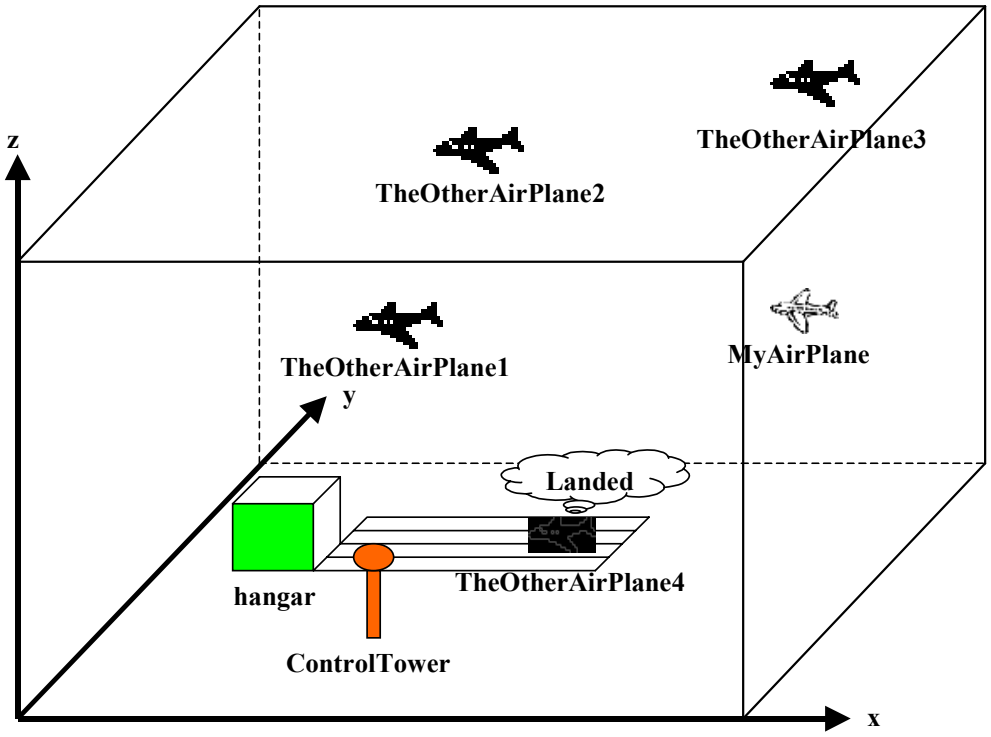
4.1. 목적 응용프로그램

이 논문에서는 TMO 모델이 실제 분산 환경에서의 실시간 응용프로그램에 어떻게 적용 되는지를 알아보기 위해서 항공기 착륙 전산 모사 시스템을 TMO 모델을 이용하여 구현하여 보았다. 이미 2절에서 언급한대로 항공기 착륙 전산 모사 시스템은 스케일 되지 않은 실시간 응용 프로그램의 대표적인 예라고 할 수 있다.

4.2. 목적 응용 프로그램의 TMO 모델

그림 4는 목적 응용프로그램의 모델을 보여준다. 우선, 모델링을 수행하기 전에 응용 프로그램에서의 대상이 되는 3차원의 공간을 설정한다. 이 공간은 실제 시스템의 전체 공간을 전산 모사 하기는 불가능하므로 실제로 관심이 있는 대상 공간에 대해서만 모델링을 수행하는 것이다. 이 공간 내에는 사용자가 조정하는, 즉 MyAirPlane이라고 불리는 파일럿이 훈련에서 조정할 항공기가 있어야 하고, TheOtherAirPlane이라고 불리우는 다른 항공기가 있어야 한다. 다른 항공기는 상황에 따라서 그 수가 바뀔 수 있다. 마지막으로 공항에는 격납고, 관제탑, 활주로등의 객체가 있어야 한다.

모델링 방법은 top-down의 접근 방식을 가지고 모델링 하였다. 우선



Space

그림 4 항공기 착륙 전산모사의 모델

그림 4처럼 전체 대상을 하나의 객체로 모델링 한 후에 MyAirPlane이나 ControlTower같은 각각의 객체로 분리 가능한 것들을 각각의 객체로 분리해서 각각을 모델링을 수행한다. 이 방식으로 TMO가 응용 프로그램에 적당하게 나뉘어 질 때까지 반복적으로 수행하여 하나의 TMO를 여러개의 TMO로 나누게 된다. 본 논문에서는 아래에서 설명하는대로 모두 4개의 TMO 객체로 나누어 모델링을 수행했다. 각각의 모델에는 그 객체의 이름과, TMO에 의해서 유지/갱신 되는 정보를 유지하는 ODS와 TMO객체간의 통신을 위한 접근 가능한 객체들의 목록, 실시간으로 매 주기마다 수행되는 SpM들, 그리고 외부 TMO로부터 메시지를 받아 들이는 SvM들로 구성된다.

1) MyAirPlaneTMO : 사용자에 의해서 조정되는 항공기를 모델링한다. 이 객체는 크게 3가지 종류의 정보를 가진다. i) 속도, 위치같은 항공기의 상태(states), ii) 크기, 최대 속도 같은 항공기의 특성(properties), iii) 사용자가 비행기를 조정하므로 사용자 입력에 대한 방향타, 보조날개, 착륙 장치 등의 상태에 관한 정보(user control) 등이 필요하다. 이러한 정보들은 TMO 객체 내의 ODS영역에 저장되어 진다. 또한 이 객체는 ControlTowerTMO라고 불리는 관제탑 객체와 통신하기 위한 채널이 있어야한다. 이 통신 채널은 사용자가 관제탑에 활주로를 요청하는 것과 관제탑에서 다른 비행기등과의 충돌에 대한 경고 메시지를 받는 등에 사용된다. 뿐만 아니라 기상에 따라서 비행기의 상태가 영향을 받으므로 기상 정보를 받기 위한 SpaceTMO와의 통신 채널이 있어야 한다. 그리고 MyAirPlaneTMO에는 비행기의 상태를 다음 상태로 갱신하는 UpdateStates SpM이 있어야 하고, 사용자의 입력을 처리하는 ProcessingUserControl SpM이 있어야 한다. 그림 5는 이렇게 모델링한 결과를 보여준다. ODS로 사용자의 입력(UserControl), 속성(Properties), 상태(States), SpaceTMO로부터 받은 기후정보의 복사본을 가지고 있고, 위에서

MyAirPlane
Access Capability : Space, ControlTower
ODS - UserControl // flaps, rudder, landing gears, spoilers, etc. - Properties // mileage, weight, max speed, lower bound speed to fly, etc. - States // velocity, position, remaining fuel, views, flight distance, alarm, balances, etc. - Copy_Weather_From_Space

SpMs
- Update States
// Update velocity, position, current weight, remaining fuel, balances, etc.
- Processing User Control
// Update flaps, rudder, landing gear, request landing runway to ControlTower, etc.

SvMs
- ReceiveFromSpace
// Get Weather Info. from Space.
- ReceiveFromControlTower
// Get result of requesting for landing, alarm message, etc.

그림 5 MyAirPlane 객체의 TMO 모델.

설명한대로 UpdateStates SpM과 ProcessingUserControl SpM이 있게 된다. 또 SpaceTMO로부터 들어오는 메시지를 처리하기 위한 ReceiveFromSpace SvM과 관제탑으로부터 들어오는 메시지를 처리하기 위한 ReceiveFromControlTower SvM이 있게 된다.

2) SpaceTMO : 이 객체는 3차원의 전산 모사를 수행하는 공간을 나타낸다. 이 객체는 전산 모사에 필요한 모든 정보를 가지고 있으면서 TheOtherAirPlane의 상태를 갱신하고, 기타 기후 등의 상태를 갱신하는 SpM

Space
Access Capability : ExternalSpace, ControlTower, MyAirPlane
ODS - Signature_MyAirPlane - (0 – N) TheOtherAirPlanes - (0 – M) Runways - Weather_and_Others

SpMs
- UpdateTheOtherAirPlanes
// Change the state of TheOtherAirPlanes(something like position, velocity…etc)
// and push the data to ExternalSpace or Hangar.
- UpdateWeather_and_Others
// Wind, humidity, rain, snow, air pressure, etc and push to MyAirPlane.

SvMs
- ReceiveFromExternalSpace
// Receive new flying airplane and include it to TheOtherAirPlanes.
- ReceiveFromMyAirPlane
// Receive the information such as position, velocity, and so on, from MyAirPlane

그림 6 Space 객체의 TMO모델.

들을 갖어야 한다. 그러므로 그림 6에서 보는 바와 같이 UpdateTheOtherAirPlanes SpM과 UpdateWeather_and_Others SpM이 있게 된다. 또한 SpaceTMO는 전산 모사를 수행하고자 하는 공간외에서 들어오는 TheOtherAirPlane과, 공간내에서 생성되는 TheOtherAirPlane에 대한 처리를 수행하기 위해서 ExternalSpace TMO로부터 받아들이는 채널이 필요하게 된다. 또한 실제 관제탑은 MyAirPlane과의 통신을 통해서가 아니라 직접 SpaceTMO를 통해서 TheOtherAirPlane의 정보들 뿐만 아니라 MyAirPlane

ControlTower
Access Capability : Space, MyAirPlane
ODS - Properties // position, size - ControlSystem // Computer system for managing airport
SpMs - UpdateControlSystem // Using all data of airplanes, check the danger of crash for MyAirPlane and report it.

SvMs
- ReceiveFromMyAirPlane
// Processing request from MyAirPlane
- ReceiveFromSpace
// Receive all airplanes info from Space

그림 7 ControlTower 객체의 TMO 모델.

의 정보를 받을 수 있어야 한다. 그러므로 MyAirPlaneTMO에서 SpaceTMO에게 MyAirPlane의 정보를 줄 수 있는 통신 채널이 필요하다. 그러므로 SvM으로 ReceiveFromMyAirPlane이 필요하게 된다.

3) ControlTower TMO : 전산 모사를 수행하는 공간내에 공항이 존재 하므로 그 공항을 관제하는 관제탑을 모델링 하는 객체가 필요하게 된다. 그래서 ControlTower TMO가 필요하게 된다. 이 객체가 하는 일은 공간 내의 비행기들의 상태나 주변 환경등의 정보를 가지고 항공기들을 관제 하는 일

ExternalSpace
Access Capability : Space
ODS - NewAirPlane // new flying airplane for entering space - RecievedAirPlane // old flying airplane from space which must be removed
SpMs - UpdateNewAirPlane // Randomly generate new flying airplane, and push it to Space.
SvMs - ReceiveFromSpace // Receive old flying airplane(which get out from Space) and remove it.

그림 8 ExternalSpace 객체의 TMO 모델.

을 하게 된다. 그림 7에서 보는 바와 같이 이 객체는 자신의 위치와 크기 같은 속성 정보와 실제로 주변을 관제하기 위한 관제 시스템을 ODS로 가지고 있어야 한다. 또한 관제 시스템의 상태를 매 단계마다 갱신해 주기 위한 UpdateControlSystem이라는 SpM을 가지고 있어야 하고, SvM으로는 MyAirPlaneTMO와 통신을 하기 위한 ReceiveFromMyAirPlane SvM과 SpaceTMO로부터 정보를 받아 오기 위한 ReceiveFromSpace SvM이 있어야 한다.

4) ExternalSpaceTMO : 이 객체는 전산 모사 대상이 아닌 그 외부

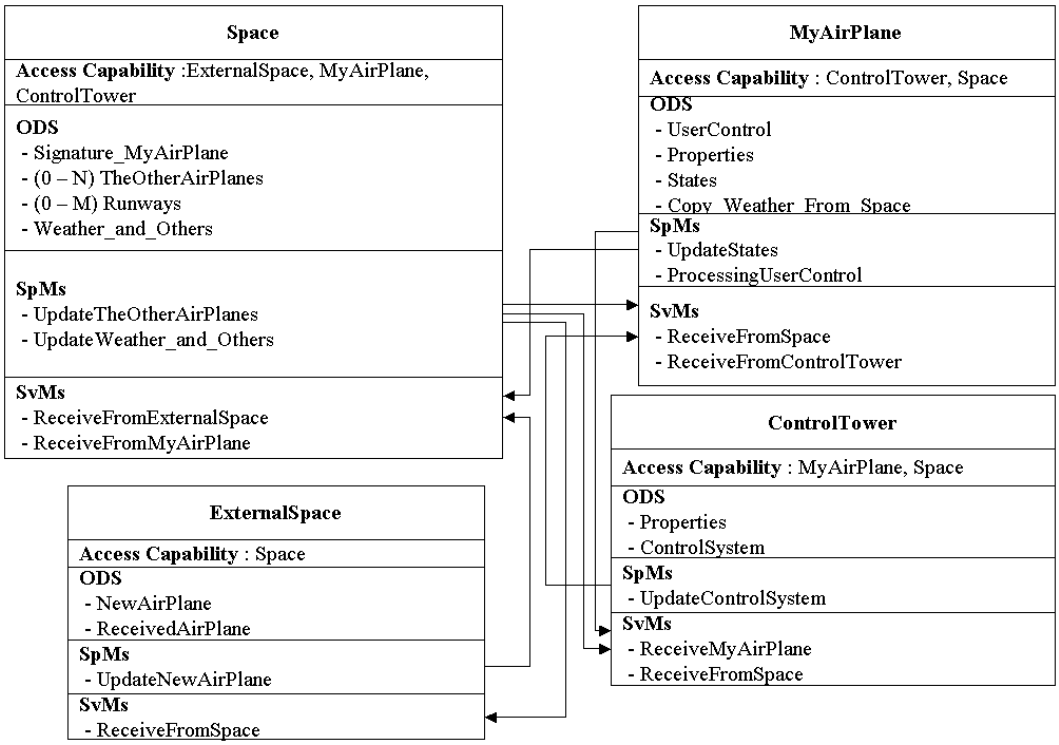


그림 9 TMO 객체간의 상호관계.

의 공간에 대한 모델이다. SpaceTMO가 관장하는 공간은 실제 공간의 극히 일부분에 불과하다. 그 외의 공간에 대해서 전산 모사를 수행하는 것은 아무런 의미가 없다. 그러므로 SpaceTMO가 관장하는 이외의 공간은 어떠한 다른 방법으로 모델링을 하여야 한다. 그래서 여기서는 외부 공간을 다른 비행기를 생성하고, 소멸 시키는 대상으로 모델링을 수행하였다. 그림 8은 이렇게 모델링을 한 ExternalSpace TMO를 보여준다. ODS로는 새로 생성되는 항공기

SpaceTMO에서 공간을 이탈해서 더 이상 전산 모사의 관심의 대상이 아닌 항공기를 받아들이는 ODS를 가지게 된다. SpM으로는 어떠한 특정한 방법에 의해서 새로이 비행기를 생성 시키는 UpdateNewAirPlane이 있게 된다. 이 SpM은 주기적으로 수행되므로 매 실행될 때마다 새로운 항공기를 생성하는 것이 아니라 특수한 경우에 비행기를 생성하도록 한다. 그러나 본 논문에서는 그 특정 알고리즘은 중요하지 않으므로 랜덤하게 비행기가 생성 되도록 하였다. 그리고 SpaceTMO의 공간을 이탈한 비행기를 처리하기 위한 ReceiveFromSpace라는 SvM이 있게 된다.

이렇게 모델링된 각각의 객체들의 상호 관계는 그림 9에 나타내었다. 각각의 객체간의 화살표는 메시지가 전송 되는 방향을 나타낸다. 앞에서 이미 설명되었던과 마찬가지로 각각의 TMO 객체는 서로 다른 TMO객체와 통신을 하도록 되어있는 것을 볼 수 있다.

5. 구현

5.1. 시각 단위(Time Unit)

실시간 전산 모사의 성능은 전산 모사에 사용된 시각 단위라고 할 수 있다. 실시간 전산모사의 시각 단위란 전산 모사 시스템의 클럭이 일정하게 발생되는 간격을 이야기한다. 각각의 클럭이 발생되는 간격 사이에는 전산모사가 한 단계씩 진행하게 된다. 실시간 전산 모사의 성능을 증가시키기 위해서, 즉 전산 모사의 정밀성을 보다 더 증가시키기 위해서는 전산 모사 시각 단위가 반드시 최소화 되어야 한다. 그렇다면 서로 다른 주기를 갖는 여러 개의 SpM이 있는 경우에 어떻게 시각 단위를 설정해야 하는가라는 의문이 남는다. 물론 가장 쉽고 자연스러운 방법은 각각의 주기로 전산 모사를 수행해 나가는 것이다. 그러나 이 방법은 언제 SpM이 전달하고자 하는 메시지를 목적지에 전달 시키게 되는지 알기 어렵게 하므로 문제가 될 수 있다. 예를 들어 2개의 TMO가 각각 하나의 SpM을 가지고 있고, 1번 노드에서는 주기가 30 ms이고 다른 노드에서는 주기가 20 ms라고 가정해 보자. 전산 모사는 매 단계별로 진행되어 나가므로 한 단계의 계산이 끝나면 그 결과를 이용하여 다음 단계의 계산을 진행해 나가게 된다. 이때 서로 다른 TMO간의 통신이 있으므로 서로 메시지를 주고받게 된다. 그림 10은 이렇게 메시지를 주고받는 것을 나타낸다. 그림 10을 보면 노드 1은 30 ms가 주기이고, 노드 2는 20 ms가 주기인 것을 보여준다. 여기서는 노드 2가 매 단계마다 하나의 메시지를 노드 1에게 전달하는 예를 보여 주고 있다. 두 노드의 전산 모사 시작 시각이 동일하면 첫 번째 단계에서는 정상적으로 노드 2의 메시지 하나가 노드 1로 전성

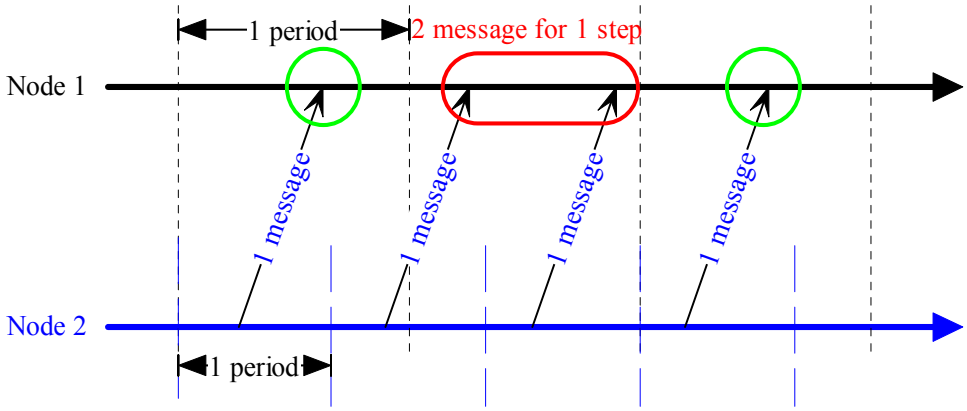


그림 10 다른 주기를 갖는 TMO 객체간 메시지 전송에서의 문제점.

되게 된다. 그러나 node 1의 주기와 node 2의 주기가 서로 다르므로 node 1의 두 번째 단계에서는 노드 2의 2개의 메시지를 받게 되는 일이 발생한다. 만약에 이렇게 2개의 메시지가 한 단계에 도착하게 되면 노드 1에서는 어떠한 값을 취해서 전산 모사를 진행시켜 나가야 할지 알 수 없게 된다. 만약 두 메시지 중에 하나의 값을 취한다면 제대로 된 값을 가지고 계산 하는 것이 아니게 되므로, 어떠한 방법을 통해서 두개의 메시지를 하나로 만들어서 (예를 들어 두 값을 평균을 이용한다) 사용하는 등의 방법이 필요하게 된다. 이는 전산 모사의 수행을 어렵게 만들고 언제 메시지가 도착하는지 예측하기 어렵게 만든다. 그러므로 전산 모사를 수행하는데 있어서 모든 시각 단위는 동일하게 유지 되어야한다. 그러므로 목적 응용프로그램을 장성할 때도 모든 전산 모사 시각단위는 동일하게 만들었다.

```
class MyAirPlaneTMO_ODSS : public ODSSBaseClass
{
private :
    int m_fuel;
    struct cartesian_vector m_vel;
    struct  cartesian_vector    m_pos;
    struct  flaps       m_flaps;
    struct  spoilers    m_spoilers;
    struct  size    m_size;
    int     m_rudder;
    int     m_landinggear;
    int     m_mileage, m_weight, m_maxspeed;
    struct  balances    m_balance;
    struct  engineThrotle   m_enginethrotle;
    struct  weather m_weather;
    void    setDefault();
    int m_AlarmReceived;
```

그림 11 MyAirPlaneTMO의 ODS중의 일부.

5.2. 항공기 착륙 전산 모사 시스템의 구현

TMO 모델로 모델링 되어 있는 시스템을 직접 구현 하는 것은 매우 쉽다. 이것은 단지 모델을 C++ 원시 코드로 변환시키기만 하면 된다. 그림 11은 이렇게 변환되어진 MyAirPlaneTMO의 ODS부분의 일부를 보여준다. ODS는 위에서 언급한대로 TMO객체가 유지/갱신 하는 객체의 상태나 속성등의 정보를 저장하는 곳이다. 그 내용을 살펴보면 우선 상태를 나타내기 위한 몇 가지 변수들이 있다. 첫 번째로 m_vel은 MyAirPlane의 속도를 나타내기 위한 변수로 cartesian_vector라는 구조체형으로 구현되어있다. cartesian_vector라는 자료형은 3차원 공간안의 좌표를 표시하기 위해서 x,

```
class MyAirPlaneTMO_UpdateState_SpM : public SpMBaseClass
{
private :
    MyAirPlaneTMO_ODSS  * m_MyAirPlaneTMO_ODSS;
    TMOGateClass    *
        m_gate_MyAirPlaneTMO_UpdateState_SpM_to_SpaceTMO_ReceiveFromMyAirPlane_SvM;
    TMOGateClass    *
        m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMyAirPlane_SvM;
    struct ParamStruct_FromMyAirPlane_to_Space  mp_toSpace;
    struct ParamStruct_FromMyAirPlane_to_ControlTower   mp_toControlTower;
public :
    MyAirPlaneTMO_UpdateState_SpM(const char * SpM_name, TMOGateClass &gate1,
        TMOGateClass & gate2, MyAirPlaneTMO_ODSS & odss, access_mode_type mode);
    ~MyAirPlaneTMO_UpdateState_SpM();
    virtual void SpMBody();
};
```

그림 12 MyAirPlane TMO의 UpdateState SpM의 선언부.

y, z로 표현되는 공간내의 좌표를 표시할 수 있는 구조체이다. 속도는 벡터로 표현될 수 있으므로 cartesian_vector 자료형을 이용하여 선언되었다. 이 속도는 다음 상태에서의 비행기의 위치를 계산하는데 사용되고, 다음 단계의 항공기의 속도를 계산하는 기초가 된다. 다음 변수로는 역시 cartesian_vector 형태로 선언된 m_pos가 있다. 이것은 공간내의 MyAirPlane의 위치를 나타내는 변수이다. 다음의 flaps라는 구조체로 선언된 m_flaps는 보조날개의 상태를 나타내 주는 변수이고, m_spoiler는 항공기의 속도를 저하시킬 때 사용하는 스포일러의 상태를 나타내 주는 변수이다. 또한 m_rudder는 항공기의 방향타를 나타내 주는 변수이고 m_landinggear는 항공기의 착륙장치의 상태를 나타낸다. 또한 engineThrotle이라는 구조체로 선언된 m_enginethrotle이라는 변수는 엔진의 출력을 나타내기 위해서 사용된 변수로서 항공기의 다음 단계에서의 속도를 계산하는데 사용된다. 또 m_mileage나 m_weight, m_maxspeed 같은 변수들은 변수 이름에서 그 역할이 나타나므로 자세한 설

명은 생략 하도록 한다. 이러한 모든 내용들은 앞의 4절에서 모델링한 것들을 각각 다 실제 C++ 원시 코드로 단순히 변환 시킨 것이다. 이 모든 ODS의 값들은 TMO 객체 내의 SpM이나 SvM에 의해서 유지/갱신되어 진다.

그림 12는 MyAirPlaneTMO의 UpdateState SpM의 클래스 선언부를 보여준다. 이 클래스의 이름은 MyAirPlaneTMO_UpdateState_SpM이고, 매 주기마다 한번씩 수행된다. 일반적으로 실제 시스템의 모델링을 수행하면 각각 서로 다른 주기를 갖는 몇 개의 SpM으로 나뉜다. 마찬가지로 앞의 4절에서 모델링을 한 목적 응용프로그램의 객체들도 역시 여러 개의 서로 다른 주기를 갖을 수 있는 SpM들로 모델링 되었다. 그러나 위의 6.1절에서 언급한대로 모든 SpM들은 동일한 시각 단위, 즉 주기를 갖는 것이 좋으므로 모든 TMO 객체의 SpM들의 주기는 동일하게 구현하였다. 그림 12에서 보여주는 SpM이 하는 일은 SpM의 이름 자체에서 이미 알 수 있듯이 항공기의 상태를 갱신하며 다른 TMO 객체들에게 필요한 메시지(자신의 위치 정보를 SpaceTMO에게 전달한다던가, 필요한 메시지를 관제탑에게 전달하는 등)를 보낸다. 반면에 사용자의 입력은 MyAirPlane TMO내에 있는 Process_User_Control_SpM에 의해서 수행되나 논문에서는 보이지 않는다.

그림 13은 이렇게 구현되는 MyAirPlane TMO의 클래스 선언부를 나타낸다. 위에서 설명한 ODSS를 가지고 있고, 2개의 SpM을 가지고 있는 것을 볼 수 있다. 또한 다른 TMO들과 통신을 수행하기 위해서 2개의 SvM을 갖는다. 하나는 Space TMO와의 통신을 하기 위한 SvM이고 다른 하나는 ControlTower TMO와 통신을 수행하기 위한 SvM이다.

다른 객체들, 즉 ControlTower TMO나 Space TMO, ExternalSpace TMO등도 역시 위에서 예로들은 MyAirPlane TMO와 비슷한 방식으로 구현되었다. 역시 몇 개의 SpM과 SvM들로 이루어져 있으며, 각각 필요한 정보를 저장하는 ODS를 가지고 있다. 여기서도 역시 마찬가지로 모든 SpM들은 같은

```
class MyAirPlaneTMO : public TMOBaseClass
{
private :
    MyAirPlaneTMO_ODSS  m_MyAirPlaneTMO_ODSS;
    MyAirPlaneTMO_UpdateState_SpM
        m_MyAirPlaneTMO_UpdateState_SpM;
    MyAirPlaneTMO_ProcessUserControl_SpM
        m_MyAirPlaneTMO_ProcessUserControl_SpM;
    MyAirPlaneTMO_ReceiveFromSpace_SvM
        m_MyAirPlaneTMO_ReceiveFromSpace_SvM;
    MyAirPlaneTMO_ReceiveFromControlTower_SvM
        m_MyAirPlaneTMO_ReceiveFromControlTower_SvM;
public :
    MyAirPlaneTMO(const char * TMO_name, TMOGateClass &,
        TMOGateClass &, tms & TMO_start_time);
    ~MyAirPlaneTMO();
};
```

그림 13 MyAirPlaneTMO의 선언부.

시각 단위, 즉 SpM의 주기를 가지고 전산 모사를 수행해 나간다.

5.3. 사용자 응용 프로그램의 구현

TMO 객체를 포함해서 구현한 응용 프로그램은 사용자가 직접 사용하기에 적합하지 않다. 왜냐하면 TMO 응용 프로그램은 GUI(Graphic User Interface)을 지원하지 않기 때문이다. 그러므로 Microsoft Visual C++ 6.0을 이용하여 MFC(Microsoft Foundation Classes)를 사용하여 사용자 응용 프로그램을 작성하였다. TMO 응용 프로그램과 본 사용자 프로그램간의 통신은 UDP(User Datagram Protocol)을 사용하였다. 원칙적으로 사용자 응용 프로그램도 역시 실시간 응용 프로그램으로 작성 되어야 실시간 전산 모사를

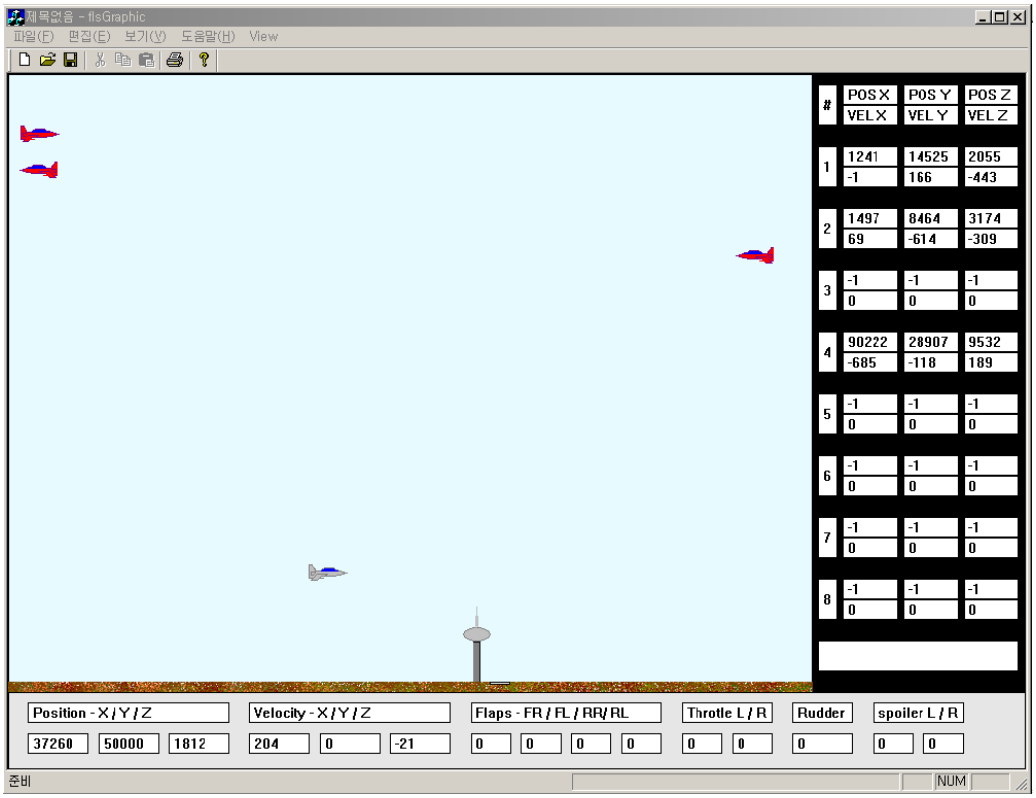


그림 14 사용자 응용 프로그램의 모습.

수행하는 목적에 적당할 것이다. 그러나 MFC나 Windows NT에서는 실시간 응용 프로그램의 작성을 지원하지 않으므로 실시간으로 작성할 수 없다. 그러므로 사용자 응용 프로그램은 일반 MFC를 이용한 프로그램으로 작성 되었다. 그러나 수 밀리 초 단위의 오차는 사용자가 인식하지 못하므로 이 논문에서 작성한 응용 프로그램에는 큰 문제가 되지 않음을 알 수 있다. 이 응용 프로

그램이 수행되는 상태는 그림 14에서 볼 수 있다.

5.4. 병렬성의 확장

TMO 모델은 분산 환경을 지원한다. 각각의 노드는 하나 혹은 그 이상의 TMO 객체를 가지고 수행시킨다. 본 논문에서 구현한 항공기 착륙 전산 모사 시스템은 4개의 TMO로 모델링을 수행하고 구현하였으므로 최대 4개 노드에서 시스템을 수행할 수 있다. 만일 더 많은 노드에서 전산 모사를 수행하고자 한다면 더 많은 TMO로 나누어서 구현하여야만 한다.

예를 들어 SpaceTMO를 살펴보자. 본 논문에서는 SpaceTMO에서 모델링 하는 전체 공간내의 기상 조건(기온, 습도, 풍속, 기압 등)은 모두 동일하다고 가정하였다. 이는 매우 좁은 공간을 모델링 하는 경우에는 적용이 가능하지만 실제의 기상 조건은 공간내의 각 위치마다 서로 다른 기상 조건을 가지게 된다. 그러므로 보다 정확한 전산 모사를 위해서는 전체 공간을 동일한 기상으로 모델링을 하는 것 보다 공간을 여러 개의 하위 공간으로 나누어서 각각의 하위 공간들이 서로 다른 기상 조건을 갖도록 구현해야 한다. 물론 이 경우에 하위 공간의 크기(일반적인 의미로의 부피)가 작아지면 작아질수록 보다 정확한 전산 모사가 된다. 그러므로 우리는 SpaceTMO를 더 많은 수의 하위 TMO로 구성할 수 있다. 이 각각의 하위 TMO들은 서로 간에 통신을 수행할 수도 있고, 다른 TMO모델들과도 통신을 할 수도 있다. 이렇게 하나의 TMO를 여러개의 하위 TMO로 구성함으로서 분산 환경을 효율적으로 사용할 수 있도록 구현할 수 있다.

6. 목적 응용프로그램의 성능

6.1. SpM의 예상 수행 시각과 실제 수행 시각의 차이

표 4는 실제 응용프로그램 즉, 5절에서 구현한 실시간 항공기 착륙 전산모사 시스템의 SpM의 예상 수행 시각과 실제 수행 시각의 차이를 보여준다. 실제 응용 프로그램이 4개 노드에서 수행되므로 4개 노드에서의 결과를 나타낸다. 표 4의 결과를 살펴보면 3.1절의 표 1에서의 결과와 매우 비슷한 것을 볼 수 있다. 최대 값은 약 9 ms이고 평균값도 역시 4.5 ms에 가까운 것을 볼 수 있다. 사용자와 연계를 통한 응용 프로그램은 수십 미리 초 단위의 시간차이는 크게 느껴지지 않는다. 그러므로 위의 9 ms의 오차는 허용가능한 오차 범위내에 있다고 할 수 있다.

표 5 실제 목적 응용 프로그램에서 SpM의 예상 수행 시각과 실제 수행 시각의 차이, 단위 : microseconds

노드 수	최소값	최대값	평균값	표준 편차
4	37	9065	4550.923	2604.452

6.2. 여러 노드간의 수행 시각의 차이

표 5는 구현한 항공기 착륙 전산모사 시스템을 여러 노드에 분산시켰을 때 각각의 노드간의 수행 시각의 차이를 보여준다. 모든 값들은 앞의 3절에서와 마찬가지로 각 노드간 수행시각 차이 중에 최대값을 선택하였다. 결과는 역시 3.2절의 표 3과 비슷한 결과를 보여준다. 이 결과는 실제 응용 프로그램에서도 TMO모델을 이용한 결과가 적합함을 보여준다. 역시 최대 결과의 오차가 9 ms 정도인 것은 사용자가 느끼기 어려운 정도의 차이이므로 사용자와의 연계를 통한 전산 모사 시스템의 구현에 적합함을 볼 수 있다. 뿐만 아니라 노드간의 차이가 평균적으로 매우 적으므로 그 성능이 매우 좋은 것을 볼 수 있다.

표 6 실제 목적 응용 프로그램에서 여러 노드에 분산되어 있는 SpM들 간의 수행 시각의 차이, 단위 : microseconds

노드 수	최소값	최대값	평균값	표준 편차
4	1	9019	24.9572	336.399

6.3. 노드 수와 성능과의 관계

이미 앞에서도 언급 하였듯이 실시간 전산 모사 응용 프로그램에서의 성능은 정해진 시간을 넘기지 않고 수행할 수 있는 최소한의 주기로 나타낼

수 있다. 특히 스케일 되지 않은 실시간 전산 모사에서는 매우 좋은 척도가 된다. 표 6에서는 구현한 실제 응용 프로그램을 한 노드에서 수행시켰을 때의 결과와 4개 노드에서 수행시켰을 때의 결과를 보여준다. 이 결과들은 모드 10,000번의 수행에 제한 시간(deadline)을 한번도 넘기지 않은 최소한의 시간을 측정한 것이다.

이 결과를 보면 4개 노드에서는 18 ms로 1개 노드의 30 ms 보다 좋은 성능을 나타내는 것을 볼 수 있다. 이러한 성능 향상의 요인은 1개 노드에서 전산 모사를 수행하는 경우 각각의 SpM들이 수행되면서 각각의 SpM에 소모되는 CPU 시간이 4개 노드로 분산되면서 각각의 노드에서는 각각의 노드가 가지고 있는 SpM들만 수행시키는 것으로 나누기 때문이다. 단일 노드에서의 결과는 주기가 30 ms이므로 사용자 연계 응용프로그램에 사용하기에는 적합한 결과라는 것을 알 수 있다. 또한 4개 노드에서의 결과는 18 ms로 보다 정확한 결과를 얻을 수 있다.

표 7 실제 응용 프로그램에서의 최소 전산 모사 시각 단위

노드 수	1개 노드	4개 노드
최소 주기	30 ms	18 ms

7. 결론

본 논문에서는 TMO 모델을 이용하여 실제 응용 프로그램을 구현 하였고, 그 성능을 평가하였다. TMO 모델에 대한 개괄적인 이해와 더불어 TMO 모델이 실제 응용 프로그램의 요구사항을 맞추어 줄 수 있는지를 평가해 보았다. 평가 결과는 수십 밀리 초 단위의 주기를 가지고 수행되는 실시간 응용 프로그램의 구현에 적합한 성능을 보이는 것을 볼 수 있었다. 뿐만 아니라 TMO 모델의 분산 환경에서의 적합성을 살펴보았다. 이 결과 역시 수십 밀리 초 단위의 주기를 가지고 수행되는 경우에는 적합함을 볼 수 있었다.

TMO 모델은 대상을 실시간 응용에 맞게 모델링 하는 하나의 방법론을 제시한다. TMO는 top-down 설계론을 기반으로 분산 환경을 지원하는 객체 지향형 모델이다. 이 방법에 따라 전산 모사를 수행할 수 있는 여러 대상 중에 스케일 되지 않은 전산 모사 대상으로 대표적인 항공기 착륙 전산 모사 시스템의 모델링을 수행하였고 그 과정 및 결과를 제시하였다. 그리고 이를 기반으로 실제 응용 프로그램을 작성하고 그 성능을 측정 하였다. 성능 측정은 전산 모사의 시각 단위를 기준으로 하였다. 동일한 부하를 걸리도록 하는 응용 프로그램에서 제한 시각을 지켜주는 최소한의 시각 단위는 그 시스템의 성능이라고 할 수 있다.

실제 응용프로그램의 성능은 앞에서 평가를 수행하였던 TMO 자체가 필요한 요구 사항을 맞추어 준 것처럼 실제 그 성능에 문제를 보이지 않았다. 4개 노드에서의 18 ms 의 주기는 사용자에게 응용 프로그램이 계속 연속적으로 진행된다고 생각할 수 있는 정도의 주기이고, 최대 오차가 10 ms정도인 것도 역시 사용자가 느끼기 힘든 정도의 오차이다. 이는 TMO 모델과 그

구현이 실제로 사용자 연계를 하는 응용 프로그램을 작성하는데 적합함을 보여준다.

TMO는 분산 환경을 지원하므로 실제 분산 환경에서의 적합성도 역시 실험하였다. 이를 위해서 각각의 TMO를 여러 노드에 분산시켜 수행시켜 보았다. 결과는 예상대로 노드 수의 증가에 따라 전산 모사 시스템의 성능이 향상되는 것을 볼 수 있었다. 그러므로 TMO 모델은 분산 환경에서의 실시간 응용 프로그램을 작성하는데 적합한 모델임을 확인 하였다.

참고 문헌

[1] K. H. (Kane) Kim, Juqiang Liu, Masaki Ishida, and Inho Kim, "Distributed Object Oriented Real-Time Simulation of Ground Transportation Networks with the TMO Structuring Scheme", *Proc. COMPSAC `99*, Phoenix, AZ, Oct. 1999, pp. 130-138.

[2] K. H. (Kane) Kim, "Object Structures for Real-Time Systems and Simulators", *IEEE Computers*, Vol. 30, No. 8, Aug. 1997, pp. 62-70.

[3] Y. Ishikawa, H. Tokuda, and C. W. Mercer, "An Object-Oriented Real-Time Programming Language", *IEEE Computer*,1992, pp. 66-73.

[4] C. W. Mercer and H. Tujuda, "The ARTS Real-Time Object Model", *Proc. 11th Real-Time System Symposium*, 1990, pp. 2-10.

[5] K. H. (Kane) Kim, "Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme", *Int'l Journal of Software Engineering and Knowledge Engineering*, Vol. 9, No. 2, 1999, pp. 251-276.

[6] K. H. (Kane) Kim, "Analysis of Guaranteed Service Times of Distributed Real-Time Objects", *Proc. ISORC 2000*, Newport Beach, CA, Mar. 2000, pp. 408-410.

[7] Kim, K.H. et al., "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. WORDS `96*, Laguna Beach, Feb. 96, pp. 59-68.

[8] 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)*, Dana Point, Oct. 1994, pp. 36-45,

[9] K. H. (Kane) Kim, Masaki Ishida, and Juqiang Liu, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. ISORC `99*, May, 1999, pp. 54-63.

[10] Min-Gu Lee and Sunggu Lee, "Implementation of a TMO-Based Real-Time Airplane Landing Simulator on a Distributed Computing Environment", *Proc. 2002 IEEE CS Workshop on Object-Oriented Real-Time Dependable Systems(WORDS),* San Diego, Jan. 2002. To be published.

부록

A. 항공기 착륙 전산모사 시스템 원시 코드

```
//
//
//          Filename    :           inc.h & inc.cpp
//          Description :           Definitions of FLS and Operator Overloading
//          Author      :           Min-Gu Lee
//

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <winsock.h>

#ifndef DEFS_
#define DEFS_

#define GRAPHIC_NODE_NAME   "cappuccino.postech.ac.kr"
#define     SPACE_X_MAX         100000
#define     SPACE_Y_MAX         100000
#define     SPACE_Z_MAX         10000

#define LANDGEAR_DOWN       0
#define LANDGEAR_UP                     1
#define TIME_UNIT           100
#define     MAX_WIND            250
#define     MAX_THROTLE                 500
#define     THROTLE_KEY_VARIANT         10
#define     CRUISE_THROTLE      500
#define     CRUISE_SPEED        212
#define     DEFAULT_AIRPRESSURE         1000
#define PHI                                 3.14159265
#define     MAX_FLAPS_UPPER   20
#define     MAX_FLAPS_LOWER   -20
#define     MAX_RUDDER_LEFT   20
#define     MAX_RUDDER_RIGHT  20
#define MAX_RUDDER 20

#define     AIRPLANE_TYPE_LANDING       1
#define     AIRPLANE_TYPE_PASS                  2
#define     AIRPLANE_TYPE_TAKEOFF       3
#define     AIRPLANE_TYPE_MYAIRPLANE    4

#define     RUNWAY_AVAILABLE 0
#define RUNWAY_OCCUPIED             1
#define     RUNWAY_UNABLE               2


#define MSG_TYPE_FROM_SPACE_TO_MYAIRPLANE_PUSH_WEATHER
1
#define
MSG_TYPE_FROM_MYAIRPLANE_TO_SPACE_PUSH_MYAIRPLANEINFO
2
//#define   SvM_PUSHAIRPLANESTOCONTROLTOWER   1
#define
MSG_TYPE_FROM_SPACE_TO_CONTROLTOWER_PUSH_AIRPLANEINFO
1

#define     ALARM_LEVEL_0       0
#define ALARM_LEVEL_1           1
#define ALARM_LEVEL_2           2
#define     ALARM_LEVEL_3       3
#define MARGIN_LEVEL1           5000
#define MARGIN_LEVEL2           3000
#define MARGIN_LEVEL3           1000

#define
MSG_TYPE_FROM_MYAIRPLANE_TO_CONTROLTOWER_REQUEST_RUNWAY
1
```

```
#define
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_REPLY_RUNWAY
2
#define
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_PUSH_ALARM
        1
#define MSG_TYPE_FROM_OUTSPACE_TO_SPACE_PUSH_NEW_AIRPLANE
1
#define MSG_TYPE_FROM_SPACE_TO_OUTSPACE_PUSH_OLD_AIRPLANE
1



#define MYAIRPLANE_MAX_STABLE_LANDING_SPEED   100
#define MYAIRPLANE_MAX_STABLE_Z_VELOCITY      30
#define MYAIRPLANE_CRASHED_OVER_MAX_STABLE_LANDING_SPEED
-1
#define MYAIRPLANE_CRASHED_OVER_MAX_STABLE_Z_VELOCITY -2
#define MYAIRPLANE_OUT_SPACE_M_X      -5
#define MYAIRPLANE_OUT_SPACE_M_Y      -6
#define MYAIRPLANE_OUT_SPACE_P_X      -7
#define     MYAIRPLANE_OUT_SPACE_P_Y  -8
#define MYAIRPLANE_OUT_SPACE_P_Z      -9
#define MYAIRPLANE_LANDED   1


#define MAX_AIRPLANES         8
#endif

#ifndef cartesian_vector_
#define cartesian_vector_

struct cartesian_vector{
        double      x;
        double      y;
        double      z;
        bool operator != (const cartesian_vector &Rhs);
        bool operator == (const cartesian_vector &Rhs);
};
```

```
#endif

#ifndef cylinderical_vector_
#define cylinderical_vector_

struct cylinderical_vector {
        double      r;
        double theta;
        double z;
};
#endif

#ifndef size_
#define size_

struct      size        {
        int         x;
        int y;
        int         z;
        bool operator!= (const size &Rhs);
        bool operator== (const size &Rhs);
};

#endif

#ifndef AirPlane_
#define AirPlane_

struct AirPlane {
        int                     id;
        struct      cartesian_vector        pos;
        struct      cartesian_vector        vel;
        struct      size        sz;
        int                     type;
        bool operator!=( const AirPlane & Rhs );
        bool operator==(const AirPlane & Rhs);
};
```

```
#endif


#ifndef weather_
#define weather_

struct weather {
        struct      cartesian_vector      wind;
        int         airpressure;
};

#endif


#ifndef engineThrotle_
#define engineThrotle_

struct engineThrotle {
        int         right;
        int left;
};

#endif

#ifndef flaps_
#define flaps_

struct flaps {
        int         front_left;
        int         front_right;
        int         rear_left;
        int         rear_right;
};

#endif

#ifndef spoilers_
#define spoilers_
```

```
struct spoilers {
        int         left;
        int         right;
};

#endif

#ifndef balances_
#define balances_

struct balances {
        int         fr;
        int         lr;
};
#endif

#ifndef ParamStruct_FromSpace_toMyAirPlane_
#define ParamStruct_FromSpace_toMyAirPlane_

struct ParamStruct_FromSpace_to_MyAirPlane {
        int         Type;
        struct weather        w;
};

#endif

#ifndef ParamStruct_FromMyAirPlane_to_Space_
#define ParamStruct_FromMyAirPlane_to_Space_

struct ParamStruct_FromMyAirPlane_to_Space {
        int                   Type;
        struct AirPlane plane;
};

#endif

#ifndef ParamStruct_FromSpace_to_ControlTower_
#define ParamStruct_FromSpace_to_ControlTower_
struct ParamStruct_FromSpace_to_ControlTower {
```

```
        int         Type;
        struct      AirPlane plane;
};
#endif

#ifndef ParamStruct_FromControlTower_to_MyAirPlane_
#define     ParamStruct_FromControlTower_to_MyAirPlane_
struct ParamStruct_FromControlTower_to_MyAirPlane {
        int Type;
        int         Alarm;
        int         Runway;
};
#endif

#ifndef ParamStruct_FromMyAirPlane_to_ControlTower_
#define ParamStruct_FromMyAirPlane_to_ControlTower_
struct ParamStruct_FromMyAirPlane_to_ControlTower {
        int Type;
};
#endif

#ifndef ParamStruct_FromOutSpace_to_Space_
#define ParamStruct_FromOutSpace_to_Space_
struct ParamStruct_FromOutSpace_to_Space {
        int Type;
        AirPlane plane;
};
#endif

#ifndef ParamStruct_FromSpace_to_OutSpace_
#define ParamStruct_FromSpace_to_OutSpace_
struct ParamStruct_FromSpace_to_OutSpace {
        int Type;
        AirPlane plane;
};
#endif


#include "inc.h"
```

```
bool
cartesian_vector::operator != (const cartesian_vector &Rhs)
{
        return ((Rhs.x != x) || (Rhs.y != y) || (Rhs.z != z));
}
bool
cartesian_vector::operator == (const cartesian_vector &Rhs)
{
        return !((Rhs.x != x) || (Rhs.y != y) || (Rhs.z != z));
}


bool
size::operator != (const size &Rhs)
{
        return ((Rhs.x != x) || (Rhs.y != y) || (Rhs.z != z));
}

bool
size::operator ==(const size &Rhs)
{
        return !((Rhs.x != x) || (Rhs.y != y) || (Rhs.z != z));
}


bool
AirPlane::operator != (const AirPlane & Rhs)
{
        return ((id != Rhs.id) || (pos != Rhs.pos) || (sz != Rhs.sz) || (type
!= Rhs.type) || (vel != Rhs.vel));
}

bool
AirPlane::operator == (const AirPlane & Rhs)
{
        return !((id != Rhs.id) || (pos != Rhs.pos) || (sz != Rhs.sz) || (type
!= Rhs.type) || (vel != Rhs.vel));
}
```

```
//
//
//          Filename    :           MyAirPlaneTMO.h & MyAirPlaneTMO.cpp
//          Description :           Class def. of MyAirPlaneTMO and its
implementation
//          Author                  :           Min-Gu Lee
//
//


#ifndef MyAirPlaneTMO_
#define MyAirPlaneTMO_

#include "TMOSLv2.h"

using namespace TMO;

#include "MyAirPlaneTMO_ODSS.h"
#include "MyAirPlaneTMO_UpdateState_SpM.h"
#include "MyAirPlaneTMO_ProcessingUserControl_SpM.h"
#include "MyAirPlaneTMO_ReceiveFromSpace_SvM.h"
#include "MyAirPlaneTMO_ReceiveFromControlTower_SvM.h"

class MyAirPlaneTMO : public TMOBaseClass
{
private :
          MyAirPlaneTMO_ODSS m_MyAirPlaneTMO_ODSS;
          MyAirPlaneTMO_UpdateState_SpM
m_MyAirPlaneTMO_UpdateState_SpM;
          MyAirPlaneTMO_ProcessingUserControl_SpM
m_MyAirPlaneTMO_ProcessingUserControl_SpM;
          MyAirPlaneTMO_ReceiveFromSpace_SvM
m_MyAirPlaneTMO_ReceiveFromSpace_SvM;
          MyAirPlaneTMO_ReceiveFromControlTower_SvM
m_MyAirPlaneTMO_ReceiveFromControlTower_SvM;
public :
          MyAirPlaneTMO(const char * TMO_name, TMOGateClass &,
TMOGateClass &, tms & TMO_start_time);
};

#endif


#include "TMOSLv2.h"
#include "MyAirPlaneTMO.h"
#include "SpaceTMO.h"
#include "ControlTowerTMO.h"
#include "OutSpaceTMO.h"
#include "winsock.h"
#include "time.h"
#include "stdlib.h"

MyAirPlaneTMO::MyAirPlaneTMO(const char * TMO_name, TMOGateClass
&gate2, TMOGateClass & gate4, tms& TMO_start_time) :

m_MyAirPlaneTMO_UpdateState_SpM("MyAirPlaneTMO_UpdateState_SpM", gate2,
gate4, m_MyAirPlaneTMO_ODSS, RW),

m_MyAirPlaneTMO_ProcessingUserControl_SpM("MyAirPlaneTMO_ProcessingUse
rControl_SpM", gate2, gate4, m_MyAirPlaneTMO_ODSS, RW),

m_MyAirPlaneTMO_ReceiveFromSpace_SvM("MyAirPlaneTMO_ReceiveFromSpace
_SvM", m_MyAirPlaneTMO_ODSS, RW),

m_MyAirPlaneTMO_ReceiveFromControlTower_SvM("MyAirPlaneTMO_ReceiveFro
mControlTower_SvM", m_MyAirPlaneTMO_ODSS, RW)
{
          activate (TMO_name, TMO_start_time);
}

// Global Variables for UDP comm. to graphic node
int       sock, n, server_len;
struct    sockaddr_in          server, client;
struct hostent *host;
```

```
void main()
{

	// Enabling Windows Socket
	const char on = 1;
	WSADATA WSStartData;
	WSAStartup(MAKEWORD(1,1), &WSStartData);
	// initialize random seed
	srand( (unsigned)time( NULL ) );

	// Open UDP socket to graphic node
	if(!(host=gethostbyname(GRAPHIC_NODE_NAME))) {
		fprintf(stderr, "Error to get host name₩n");
		exit(2);
	}
	server.sin_family = AF_INET;
	memcpy(&server.sin_addr, host->h_addr, host->h_length);
	server.sin_port = htons(2000);
	if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
		fprintf(stderr, "client socket₩n");
		exit(3);
	}
	client.sin_family = AF_INET;
	client.sin_addr.s_addr = htonl(INADDR_ANY);
	client.sin_port = htons(0);

	if(bind(sock, (struct sockaddr *) &client, sizeof(client)) <0) {
		fprintf(stderr, "client bind");
		exit(4);
	}

	// Start TMO engine and Initialize TMOs
	StartTMOengine();
	tms		TMO_start_time1 = tm4_DCS_age(5 * 1000 * 1000);
	TMOGateClass gate1("MyAirPlaneTMO",
"MyAirPlaneTMO_ReceiveFromSpace_SvM", tm4_DCS_age(5*1000*1000));
	TMOGateClass gate2("SpaceTMO",
"SpaceTMO_ReceiveFromMyAirPlane_SvM", tm4_DCS_age(5*1000*1000));
	TMOGateClass gate3("ControlTowerTMO",
"ControlTowerTMO_ReceiveFromSpace_SvM", tm4_DCS_age(5*1000*1000));
	TMOGateClass gate4("MyAirPlaneTMO",
"MyAirPlaneTMO_ReceiveFromControlTower_SvM", tm4_DCS_age(5 * 1000 *
1000));
	TMOGateClass gate5("ControlTowerTMO",
"ControlTowerTMO_ReceiveFromMyAirPlane_SvM", tm4_DCS_age(5 * 1000 *
1000));
	TMOGateClass gate6("SpaceTMO",
"SpaceTMO_ReceiveFromOutSpace_SvM", tm4_DCS_age(5 * 1000 * 1000));
	TMOGateClass gate7("OutSpaceTMO",
"OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM", tm4_DCS_age(5*1000*1000));
	MyAirPlaneTMO		MyAirPlane("MyAirPlaneTMO", gate2, gate5,
TMO_start_time1);
	SpaceTMO Space("SpaceTMO", gate1, gate3, gate7,
TMO_start_time1);
	ControlTowerTMO	ControlTower("ControlTowerTMO", gate4,
TMO_start_time1);
	OutSpaceTMO		OutSpace("OutSpaceTMO", gate6,
TMO_start_time1);
	MainThrSleep();
}




//
//
//		Filename	:		MyAirPlaneTMO_ODSS.h &
MyAirPlaneTMO_ODSS.cpp
//		Description :		Class def. of MyAirPlaneTMO_ODSS and its
implementation
//		Author			:		Min-Gu Lee
//
//

#ifndef MyAirPlaneTMO_ODSS_
#define MyAirPlaneTMO_ODSS_

#include "TMOSLv2.h"
```

```
#include "inc.h"

using namespace	TMO;

class MyAirPlaneTMO_ODSS : public ODSSBaseClass
{
private :
		int m_fuel;
		struct cartesian_vector m_vel;
		struct	cartesian_vector	m_pos;
		struct	flaps			m_flaps;
		struct	spoilers	m_spoilers;
		struct	size		m_size;
		int			m_rudder;
		int			m_landinggear;
		int			m_mileage, m_weight, m_maxspeed;
		struct	balances	m_balance;
		struct	engineThrotle		m_enginethrotle;
		struct	weather		m_weather;
		void		setDefault();
		int m_AlarmReceived;

public :
		int m_State;
		void setAlarmState(int alarm);
		struct weather getWeather();
		void setWeather(struct weather w);

		struct engineThrotle getEngineThrotle();
		void setEngineThrotle(struct engineThrotle throtle);
		int getFuel();
		void setFuel(int fuel);
		void updateFuel();
		void updateVelocity();
		MyAirPlaneTMO_ODSS();
		void setVelocity(struct cartesian_vector vel);
		struct cartesian_vector getVelocity();
		void setPosition(struct cartesian_vector pos);
		struct cartesian_vector getPosition();
		void setFlaps (struct flaps fl);
		struct flaps getFlaps();
		void setSpoilers (struct spoilers sp);
		struct spoilers getSpoilers();
		void setRudder(int rudder);
		int		getRudder();
		void setLandingGear(int landgear);
		int		getLandingGear();
		void setProperties(int mileage, int weight, int maxspeed);
		int getMileage();
		int getWeight();
		int getMaxspeed();
		void setBalances(struct balances bal);
		struct balances getBalances();
		int updatePosition();

protected:
		BOOL m_Outside;
		char m_key;
		struct cartesian_vector toCartesian_vector(struct cylinderical_vector
vector);
		struct cylinderical_vector toCylinderical_vector(struct
cartesian_vector vector);

};

#endif


#include <math.h>
#include "TMOSLv2.h"
#include "MyAirPlaneTMO_ODSS.h"

MyAirPlaneTMO_ODSS::MyAirPlaneTMO_ODSS()
{
		setDefault();
}

void
```

```
MyAirPlaneTMO_ODSS::setDefault()
{
        EnterODSS_RW();
        m_balance.fr = 0;
        m_balance.lr = 0;
        m_spoilers.left = 0;
        m_spoilers.right = 0;
        m_vel.x = 150;
        m_vel.y = 0;
        m_vel.z = 0;
        m_pos.x = 0;
        m_pos.y = 50000;
        m_pos.z = 5000;
        m_mileage = 0;
        m_weight = 1000;
        m_rudder = 0;
        m_maxspeed = 300;
        m_flaps.front_left = 0;
        m_flaps.front_right = 0;
        m_flaps.rear_left = 0;
        m_flaps.rear_right = 0;
        m_fuel = 50000;
        m_enginethrotle.left = 200;
        m_enginethrotle.right = 200;
        m_landinggear = 0;
        m_weather.airpressure = 1000;
        m_weather.wind.x = 0;
        m_weather.wind.y = 0;
        m_weather.wind.z = 0;
        m_Outside = FALSE;
        m_State = 0;
        ExitODSS_RW();
}

struct balances
MyAirPlaneTMO_ODSS::getBalances()
{
        return m_balance;
}

struct flaps
MyAirPlaneTMO_ODSS::getFlaps()
{
        return m_flaps;
}

int
MyAirPlaneTMO_ODSS::getLandingGear()
{
        return m_landinggear;
}

int
MyAirPlaneTMO_ODSS::getMaxspeed()
{
        return m_maxspeed;
}

int
MyAirPlaneTMO_ODSS::getMileage()
{
        return m_mileage;
}

struct cartesian_vector
MyAirPlaneTMO_ODSS::getPosition()
{
        return m_pos;
}

int
MyAirPlaneTMO_ODSS::getRudder()
{
        return m_rudder;
}

struct spoilers
MyAirPlaneTMO_ODSS::getSpoilers()
```

```
{
        return m_spoilers;
}

struct cartesian_vector
MyAirPlaneTMO_ODSS::getVelocity()
{
        return m_vel;
}

int
MyAirPlaneTMO_ODSS::getWeight()
{
        return m_weight;
}

void
MyAirPlaneTMO_ODSS::setBalances(struct balances bal)
{
        EnterODSS_RW();
        m_balance = bal;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setFlaps(struct flaps fl)
{
        EnterODSS_RW();
        m_flaps = fl;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setLandingGear(int landgear)
{
        EnterODSS_RW();
        m_landinggear = landgear;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setPosition(struct cartesian_vector pos)
{
        EnterODSS_RW();
        m_pos = pos;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setProperties(int mileage, int weight, int maxspeed)
{
        EnterODSS_RW();
        m_mileage = mileage;
        m_weight = weight;
        m_maxspeed = maxspeed;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setRudder(int rudder)
{
        EnterODSS_RW();
        m_rudder = rudder;
        TMOSLprintf("RUDDER IS %d, %d₩n", m_rudder, rudder);
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setSpoilers(struct spoilers sp)
{
        EnterODSS_RW();
        m_spoilers = sp;
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setVelocity(struct cartesian_vector vel)
{
```

```
	EnterODSS_RW();
	m_vel = vel;
	ExitODSS_RW();
}

int
MyAirPlaneTMO_ODSS::updatePosition()
{
	if(!m_Outside) {
		EnterODSS_RW();
		m_pos.x = m_pos.x +  m_vel.x * TIME_UNIT / 1000;
		m_pos.y = m_pos.y +  m_vel.y * TIME_UNIT / 1000;
		m_pos.z = m_pos.z +  m_vel.z * TIME_UNIT / 1000;
		ExitODSS_RW();

		if ( m_pos.x <= 0) {
			m_Outside = TRUE;
			m_pos.x = 0;
			TMOSLprintf("Your plane is out of space
₩n");
			m_State = MYAIRPLANE_OUT_SPACE_M_X;
		} else if (m_pos.x >= SPACE_X_MAX ) {
			m_pos.x = SPACE_X_MAX;
			m_Outside = TRUE;
			TMOSLprintf("Your plane is out of space
₩n");
			m_State = MYAIRPLANE_OUT_SPACE_P_X;
		}
		if ( m_pos.y <= 0) {
			m_Outside = TRUE;
			m_pos.y = 0;
			TMOSLprintf("Your plane is out of space
₩n");
			m_State = MYAIRPLANE_OUT_SPACE_M_Y;
		} else if (m_pos.y >= SPACE_Y_MAX ) {
			m_Outside = TRUE;
			m_pos.y = SPACE_Y_MAX;
			TMOSLprintf("Your plane is out of space
₩n");
			m_State = MYAIRPLANE_OUT_SPACE_P_Y;
		}
		if ( m_pos.z <= 0) {
			m_pos.z = 0;
			/*
			if (sqrt(pow(m_vel.x, 2) +  pow(m_vel.y,2) +
pow(m_vel.z, 2)) > MYAIRPLANE_MAX_STABLE_LANDING_SPEED) {
				TMOSLprintf("Plane is crashed
!!!!₩n");
				m_Outside = TRUE;
				m_State =
MYAIRPLANE_CRASHED_OVER_MAX_STABLE_LANDING_SPEED;
			} else if (abs(m_vel.z) <=
MYAIRPLANE_MAX_STABLE_Z_VELOCITY) {
					TMOSLprintf("Plane is
crashed !!!!₩n");
					m_Outside = TRUE;
					m_State =
MYAIRPLANE_CRASHED_OVER_MAX_STABLE_Z_VELOCITY;
			} else {
				m_Outside = TRUE;
				TMOSLprintf("Your plane is
Landed ₩n");
				m_State =
MYAIRPLANE_LANDED;
			}
			*/

		} else if (m_pos.z >= SPACE_Z_MAX ) {
			m_Outside = TRUE;
			m_pos.z = SPACE_Z_MAX;
			TMOSLprintf("Your plane is out of space
₩n");
			m_State =  MYAIRPLANE_OUT_SPACE_P_Z;
		}
		return m_State;
	}
	return m_State;
}
```

```
void
MyAirPlaneTMO_ODSS::updateVelocity()
{
        if(!m_Outside) {
                EnterODSS_RW();
// Checking Rudder, rotation airplane
                m_vel.x = m_vel.x * cos(PHI * m_rudder / (2 * 1800)) -
m_vel.y * sin(PHI * m_rudder / (2 * 1800));
                m_vel.y = m_vel.x * sin(PHI * m_rudder / (2 * 1800)) +
m_vel.y * cos(PHI * m_rudder / (2 * 1800));

// checking flaps, rotation airplane
                m_vel.x = m_vel.x * cos(PHI * (m_flaps.front_left +
m_flaps.front_right + 0.3 * m_flaps.rear_left + 0.3 * m_flaps.rear_right) / (2 *
1800)) - m_vel.z * sin(PHI * (m_flaps.front_left + m_flaps.front_right +
m_flaps.rear_left + m_flaps.rear_right) / (2 * 1800));
                m_vel.z = m_vel.x * sin(PHI * (m_flaps.front_left +
m_flaps.front_right + 0.3 * m_flaps.rear_left + 0.3 * m_flaps.rear_right) / (2 *
1800)) + m_vel.z * cos(PHI * (m_flaps.front_left + m_flaps.front_right +
m_flaps.rear_left + m_flaps.rear_right) / (2 * 1800));

                m_vel.x = m_vel.x + (m_vel.x / (sqrt(pow(m_vel.x, 2) +
pow(m_vel.y, 2) + pow(m_vel.z, 2)))) * ((m_enginethrotle.left +
m_enginethrotle.right) / CRUISE_THROTLE - 0.5 + ((m_landinggear * 0.05 +
m_weather.wind.x / MAX_WIND - 0.1 * (m_spoilers.left + m_spoilers.right))) *
m_weather.airpressure / DEFAULT_AIRPRESSURE) * TIME_UNIT / 1000;
                m_vel.y = m_vel.y + (m_vel.y / (sqrt(pow(m_vel.x, 2) +
pow(m_vel.y, 2) + pow(m_vel.z, 2)))) * ((m_enginethrotle.left +
m_enginethrotle.right) / CRUISE_THROTLE - 0.5 + ((m_landinggear * 0.05 +
m_weather.wind.y / MAX_WIND - 0.1 * (m_spoilers.left + m_spoilers.right))) *
m_weather.airpressure / DEFAULT_AIRPRESSURE) * TIME_UNIT / 1000;
                m_vel.z = m_vel.z + (m_vel.z / (sqrt(pow(m_vel.x, 2) +
pow(m_vel.y, 2) + pow(m_vel.z, 2)))) * ((m_enginethrotle.left +
m_enginethrotle.right) / CRUISE_THROTLE - 0.5 + ((m_landinggear * 0.05 +
m_weather.wind.z / MAX_WIND - 0.1 * (m_spoilers.left + m_spoilers.right))) *
m_weather.airpressure / DEFAULT_AIRPRESSURE) * TIME_UNIT / 1000;

// gravity
                m_vel.z = m_vel.z + 9.8 * (sqrt( pow(m_vel.x, 2) +
pow(m_vel.y, 2) ) / CRUISE_SPEED - 1) * TIME_UNIT / 1000;
                ExitODSS_RW();
        }
}

void
MyAirPlaneTMO_ODSS::updateFuel()
{
        EnterODSS_RW();
        m_fuel -= ((m_enginethrotle.left + m_enginethrotle.right) /
MAX_THROTLE + 1);
        ExitODSS_RW();
}

void
MyAirPlaneTMO_ODSS::setFuel(int fuel)
{
        EnterODSS_RW();
        m_fuel = fuel;
        ExitODSS_RW();
}

int
MyAirPlaneTMO_ODSS::getFuel()
{
        return m_fuel;
}

struct cartesian_vector
MyAirPlaneTMO_ODSS::toCartesian_vector(struct cylinderical_vector vector)
{
        struct cartesian_vector n_vector;
        n_vector.x = vector.r * cos(asin(vector.z/vector.r)) *
cos(vector.theta);
        n_vector.y = vector.r * cos(asin(vector.z/vector.r)) *
sin(vector.theta);
        n_vector.z = vector.z;
        return n_vector;
```

```
}

struct cylinderical_vector
MyAirPlaneTMO_ODSS::toCylinderical_vector(struct cartesian_vector vector)
{
        struct cylinderical_vector n_vector;
        n_vector.r = sqrt(pow(vector.x, 2) +  pow(vector.y, 2) +
pow(vector.z, 2));
        n_vector.theta = asin(vector.y / sqrt(pow(vector.x, 2) +
pow(vector.y, 2)));
        n_vector.z = vector.z;
        return n_vector;
}

void
MyAirPlaneTMO_ODSS::setEngineThrotle(engineThrotle throtle)
{
        EnterODSS_RW();
        m_enginethrotle = throtle;
        ExitODSS_RW();
}

struct engineThrotle
MyAirPlaneTMO_ODSS::getEngineThrotle()
{
        return m_enginethrotle;
}

void
MyAirPlaneTMO_ODSS::setWeather(weather w)
{
        EnterODSS_RW();
        m_weather = w;
        ExitODSS_RW();
}

struct weather
MyAirPlaneTMO_ODSS::getWeather()
{
        return m_weather;
}

void MyAirPlaneTMO_ODSS::setAlarmState(int alarm)
{
        EnterODSS_RW();
        m_AlarmReceived = alarm;
        ExitODSS_RW();
}


//
//
//      Filename    :       MyAirPlaneTMO_ProcessingUserControl_SpM.h
& MyAirPlaneTMO_ProcessingUserControl_SpM.cpp
//      Description :       Class def. of
MyAirPlaneTMO_ProcessingUserControl_SpMS and its implementation
//      Author              :       Min-Gu Lee
//
//

#ifndef MyAirPlaneTMO_ProcessingUserControl_SpM_
#define     MyAirPlaneTMO_ProcessingUserControl_SpM_

#include "TMOSLv2.h"
#include "MyAirPlaneTMO_ODSS.h"

using namespace TMO;

#include "stdio.h"
#include "conio.h"


// Global Variables for socket comm. to graphic node

extern int  sock, n, server_len;
extern struct       sockaddr_in         server, client;
extern struct hostent *host;
```

```
class MyAirPlaneTMO_ProcessingUserControl_SpM : public SpMBaseClass
{
private :
        MyAirPlaneTMO_ODSS * m_MyAirPlaneTMO_ODSS;
        TMOGateClass        *
m_gate_MyAirPlaneTMO_UpdateState_SpM_to_SpaceTMO_ReceiveFromMyAirPlan
e_SvM;
        TMOGateClass        *
m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMy
AirPlane_SvM;
        struct ParamStruct_FromMyAirPlane_to_Space mp_toSpace;
        struct ParamStruct_FromMyAirPlane_to_ControlTower
mp_toControlTower;
public :
        MyAirPlaneTMO_ProcessingUserControl_SpM(const char *
SpM_name, TMOGateClass &gate1, TMOGateClass & gate2,
MyAirPlaneTMO_ODSS &odss, access_mode_type mode);
        ~MyAirPlaneTMO_ProcessingUserControl_SpM();
        virtual void SpMBody();
};

#endif


#include "MyAirPlaneTMO_ProcessingUserControl_SpM.h"

void
MyAirPlaneTMO_ProcessingUserControl_SpM::SpMBody()
{
        struct      engineThrotle throtle;
        struct flaps flap;
        int         rudder, landing_gear;
        char        tmpchar;
        struct spoilers         spoiler;
        char buf[256];
        struct cartesian_vector pos, vel;
        int         statecode;
        server_len = sizeof(server);


        while(_kbhit()) {
                tmpchar = _getch();
                if(tmpchar == 'a' || tmpchar == 'A') {
                        throtle =
m_MyAirPlaneTMO_ODSS->getEngineThrotle();
                        throtle.left += THROTLE_KEY_VARIANT;
                        throtle.right += THROTLE_KEY_VARIANT;
                        if(throtle.left > MAX_THROTLE) {
                                throtle.left = MAX_THROTLE;
                                throtle.right = MAX_THROTLE;
                        }

m_MyAirPlaneTMO_ODSS->setEngineThrotle(throtle);
                } else if(tmpchar == 'z' || tmpchar == 'Z') {
                        throtle =
m_MyAirPlaneTMO_ODSS->getEngineThrotle();
                        throtle.left -= THROTLE_KEY_VARIANT;
                        throtle.right -= THROTLE_KEY_VARIANT;
                        if(throtle.left < 0) {
                                throtle.left = 0;
                                throtle.right = 0;
                        }

m_MyAirPlaneTMO_ODSS->setEngineThrotle(throtle);
                } else if (tmpchar == 's' || tmpchar == 'S') {
                        flap = m_MyAirPlaneTMO_ODSS->getFlaps();
                        flap.front_left++;
                        flap.front_right++;
                        flap.rear_left++;
                        flap.rear_right++;
                        if(flap.front_left > MAX_FLAPS_UPPER) {
                                flap.front_left =
MAX_FLAPS_UPPER;
                                flap.front_right =
MAX_FLAPS_UPPER;
                                flap.rear_left =
```

```
MAX_FLAPS_UPPER;
                                        flap.rear_right =
MAX_FLAPS_UPPER;
                                }
                                m_MyAirPlaneTMO_ODSS->setFlaps(flap);
                        } else if (tmpchar == 'x' || tmpchar == 'X') {
                                flap = m_MyAirPlaneTMO_ODSS->getFlaps();
                                flap.front_left--;
                                flap.front_right--;
                                flap.rear_left--;
                                flap.rear_right--;
                                if(flap.front_left < MAX_FLAPS_LOWER) {
                                        flap.front_left =
MAX_FLAPS_LOWER;
                                        flap.front_right =
MAX_FLAPS_LOWER;
                                        flap.rear_left =
MAX_FLAPS_LOWER;
                                        flap.rear_right =
MAX_FLAPS_LOWER;
                                }
                                m_MyAirPlaneTMO_ODSS->setFlaps(flap);
                        } else if (tmpchar == 'd' || tmpchar == 'D') {
                                rudder =
m_MyAirPlaneTMO_ODSS->getRudder();
                                if(rudder++ > MAX_RUDDER)
                                        rudder = MAX_RUDDER;
                                m_MyAirPlaneTMO_ODSS->setRudder(rudder);
                        } else if (tmpchar == 'c' || tmpchar == 'C') {
                                rudder =
m_MyAirPlaneTMO_ODSS->getRudder();
                                if(rudder-- < -MAX_RUDDER)
                                        rudder = - MAX_RUDDER;
                                m_MyAirPlaneTMO_ODSS->setRudder(rudder);
                        } else if (tmpchar == 'g' || tmpchar == 'G') {

m_MyAirPlaneTMO_ODSS->setLandingGear(LANDGEAR_UP);
                        } else if (tmpchar == 'b' || tmpchar == 'B') {
m_MyAirPlaneTMO_ODSS->setLandingGear(LANDGEAR_DOWN);
                        } else if (tmpchar == 'f' || tmpchar == 'F') {
                                spoiler.left = 10;
                                spoiler.right = 10;

m_MyAirPlaneTMO_ODSS->setSpoilers(spoiler);
                        } else if (tmpchar == 'v' || tmpchar == 'V') {
                                spoiler.left = 0;
                                spoiler.right = 0;

m_MyAirPlaneTMO_ODSS->setSpoilers(spoiler);
                        } else if (tmpchar == 'r'  || tmpchar == 'R') {
                                mp_toControlTower.Type =
MSG_TYPE_FROM_MYAIRPLANE_TO_CONTROLTOWER_REQUEST_RUNWAY;

m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMy
AirPlane_SvM->OnewaySR(&mp_toControlTower, sizeof(mp_toSpace));
                        }
                }
}

MyAirPlaneTMO_ProcessingUserControl_SpM::MyAirPlaneTMO_ProcessingUserCo
ntrol_SpM(const char * SpM_name, TMOGateClass &gate1, TMOGateClass &
gate2, MyAirPlaneTMO_ODSS &odss, access_mode_type mode)
{
        m_MyAirPlaneTMO_ODSS = &odss;

m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMy
AirPlane_SvM = &gate2;
        build_regist_info_ODSS(m_MyAirPlaneTMO_ODSS->get_id(), mode);
        build_regist_info_SpM_name(SpM_name);

        MicroSec    from = 5;
        from *= 1000 * 1000;                                    // Start
after 5 seconds when process began
        MicroSec    until = 2 * 60 * 60;
        until *= 1000 * 1000;                                   // Finish 2
hours later
        MicroSec every = TIME_UNIT * 1000;                      // SpM1
```

```
must run every TIME_UNIT ms
        MicroSec est = 0 * 1000;

        MicroSec lst = 5 * 1000;
        MicroSec by = 10 * 1000;

        AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
        build_regist_info_AAC(AAC1);

        if(RegisterSpM() == FAIL)
                TMOSLprintf("Fail to register
MyAirPlaneTMO_UpdateState_SpM object ₩n");
        else
                TMOSLprintf("Succeed to register
MyAirPlaneTMO_UpdateState_SpM object ₩n");
}

MyAirPlaneTMO_ProcessingUserControl_SpM::~MyAirPlaneTMO_ProcessingUser
Control_SpM()
{
        ;
}



//
//
//      Filename    :
MyAirPlaneTMO_ReceiveFromControlTower_SvM.h &
MyAirPlaneTMO_ReceiveFromControlTower_SvM.cpp
//      Description :       Class def. of
MyAirPlaneTMO_ReceiveFromControlTower_SvM and its implementation
//      Author      :       Min-Gu Lee
//
//


#ifndef MyAirPlaneTMO_ReceiveFromControlTower_SvM_
#define     MyAirPlaneTMO_ReceiveFromControlTower_SvM_

#include "TMOSLv2.h"
#include "inc.h"
#include "MyAirPlaneTMO_ODSS.h"

using namespace TMO;

class MyAirPlaneTMO_ReceiveFromControlTower_SvM : public SvMBaseClass
{
private :
        MyAirPlaneTMO_ODSS*         m_MyAirPlaneTMO_ODSS;
        ParamStruct_FromControlTower_to_MyAirPlane
mp_fromControlTower;
public :
        MyAirPlaneTMO_ReceiveFromControlTower_SvM(const char *,
MyAirPlaneTMO_ODSS &, access_mode_type);
        void SvMBody();
};

#endif


#include "MyAirPlaneTMO_ReceiveFromControlTower_SvM.h"

MyAirPlaneTMO_ReceiveFromControlTower_SvM::MyAirPlaneTMO_ReceiveFromC
ontrolTower_SvM(const char * SvM_name, MyAirPlaneTMO_ODSS &odss,
access_mode_type mode)
{
        build_regist_info_SvM_name(SvM_name);
        m_MyAirPlaneTMO_ODSS = &odss;
        build_regist_info_ODSS(m_MyAirPlaneTMO_ODSS->get_id(), mode);
        build_regist_info_guranteed_completion_time(20 * 1000);
        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
MyAirPlaneTMO_ReceiveFromControlTower_SvM₩n");
        } else {
                TMOSLprintf("Succeed to register
MyAirPlaneTMO_ReceiveFromControlTower_SvM₩n");
```

```
		}
}

void
MyAirPlaneTMO_ReceiveFromControlTower_SvM::SvMBody()
{
		int		Client_RRQID;
		tmsp		timestamp;

		ReceiveSR(Client_RRQID, &mp_fromControlTower,
sizeof(mp_fromControlTower), timestamp);
		if(mp_fromControlTower.Type ==
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_PUSH_ALARM) {

m_MyAirPlaneTMO_ODSS->setAlarmState(mp_fromControlTower.Alarm);
		} else if (mp_fromControlTower.Type ==
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_REPLY_RUNWAY) {
			if (mp_fromControlTower.Runway == -1) {
				TMOSLprintf("There is no runway to land,
wait and request again₩n");
			} else {

TMOSLprintf("₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩nThe
Runway Number is %d₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n₩n",
mp_fromControlTower.Runway);
			}
		}
//		sprintf(buf, "Current Pos : %f, %f, %f₩n",
mp_fromMyAirPlane.plane.pos.x, mp_fromMyAirPlane.plane.pos.y,
mp_fromMyAirPlane.plane.pos.z);
//		TMOSLprintf(buf);
		TMOSLprintf("ALARM LEVEL:  %d₩n",
mp_fromControlTower.Alarm);
}



//
//
//		Filename	:		MyAirPlaneTMO_ReceiveFromSpace_SvM.h &
MyAirPlaneTMO_ReceiveFromSpace_SvM.cpp
//		Description :		Class def. of
yAirPlaneTMO_ReceiveFromSpace_SvM and its implementation
//		Author			:		Min-Gu Lee
//
//


#ifndef MyAirPlaneTMO_ReceiveFromSpace_SvM_
#define	MyAirPlaneTMO_ReceiveFromSpace_SvM_

#include "TMOSLv2.h"
#include "inc.h"
#include "MyAirPlaneTMO_ODSS.h"

using namespace TMO;

class MyAirPlaneTMO_ReceiveFromSpace_SvM : public SvMBaseClass
{
private :
		struct ParamStruct_FromSpace_to_MyAirPlane mp_fromSpace;
		MyAirPlaneTMO_ODSS*		m_MyAirPlaneTMO_ODSS;

public :
		MyAirPlaneTMO_ReceiveFromSpace_SvM(const char * SvM_name,
MyAirPlaneTMO_ODSS &odss, access_mode_type mode);
		void SvMBody();
};

#endif


#include "MyAirPlaneTMO_ReceiveFromSpace_SvM.h"

MyAirPlaneTMO_ReceiveFromSpace_SvM::MyAirPlaneTMO_ReceiveFromSpace_Sv
M(const char * SvM_name, MyAirPlaneTMO_ODSS &odss, access_mode_type
mode)
{
		build_regist_info_SvM_name(SvM_name);
```

```
        m_MyAirPlaneTMO_ODSS = & odss;
        build_regist_info_ODSS(m_MyAirPlaneTMO_ODSS->get_id(), mode);
        build_regist_info_guranteed_completion_time(20 * 1000);

        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
MyAirPlaneTMO_ReceiveFromSpace_SvM₩n");
        } else {
                TMOSLprintf("Succeed to register
MyAirPlaneTMO_ReceiveFromSpace_SvM₩n");
        }
};

void
MyAirPlaneTMO_ReceiveFromSpace_SvM::SvMBody()
{
        int         Client_RRQID;
        tmsp        timestamp;
        ReceiveSR(Client_RRQID, &mp_fromSpace, sizeof(mp_fromSpace),
timestamp);

        if (mp_fromSpace.Type ==
MSG_TYPE_FROM_SPACE_TO_MYAIRPLANE_PUSH_WEATHER) {
                m_MyAirPlaneTMO_ODSS->setWeather(mp_fromSpace.w);
        }
};



//
//
//      Filename    :       MyAirPlaneTMO_UpdateState_SpM.h &
MyAirPlaneTMO_UpdateState_SpM.cpp
//      Description :       Class def. of
MyAirPlaneTMO_UpdateState_SpM and its implementation
//      Author              :       Min-Gu Lee
//
//
```

```
#ifndef MyAirPlaneTMO_UpdateState_SpM_
#define     MyAirPlaneTMO_UpdateState_SpM_

#include "TMOSLv2.h"
#include "MyAirPlaneTMO_ODSS.h"

using namespace TMO;

#include "stdio.h"
#include "conio.h"


// Global Variables for socket comm. to graphic node

extern int  sock, n, server_len;
extern struct       sockaddr_in         server, client;
extern struct hostent *host;



class MyAirPlaneTMO_UpdateState_SpM : public SpMBaseClass
{
private :
        MyAirPlaneTMO_ODSS * m_MyAirPlaneTMO_ODSS;
        TMOGateClass        *
m_gate_MyAirPlaneTMO_UpdateState_SpM_to_SpaceTMO_ReceiveFromMyAirPlan
e_SvM;
        TMOGateClass        *
m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMy
AirPlane_SvM;
        struct ParamStruct_FromMyAirPlane_to_Space mp_toSpace;
        struct ParamStruct_FromMyAirPlane_to_ControlTower
mp_toControlTower;
public :
        MyAirPlaneTMO_UpdateState_SpM(const char * SpM_name,
TMOGateClass &gate1, TMOGateClass & gate2, MyAirPlaneTMO_ODSS & odss,
access_mode_type mode);
        ~MyAirPlaneTMO_UpdateState_SpM();
```

```
        virtual void SpMBody();
};

#endif


#include "MyAirPlaneTMO_UpdateState_SpM.h"

void
MyAirPlaneTMO_UpdateState_SpM::SpMBody()
{
        struct      engineThrotle throtle;
        struct flaps flap;
        int         rudder, landing_gear;
        char        tmpchar;
        struct spoilers         spoiler;
        char buf[256];
        struct cartesian_vector pos, vel;
        int         statecode;
        server_len = sizeof(server);

        statecode = m_MyAirPlaneTMO_ODSS->updatePosition();
        if (statecode != 0) {
                sprintf(buf, "1c%d", statecode);
                if(sendto(sock, buf, strlen(buf), 0, (struct sockaddr *)
&server, server_len) < 0) {
                        fprintf(stderr, "Error at client sendto ");
                        closesocket(sock);
                        exit(5);
                }
        }

        m_MyAirPlaneTMO_ODSS->updateVelocity();
        m_MyAirPlaneTMO_ODSS->updateFuel();

        mp_toSpace.plane.vel = m_MyAirPlaneTMO_ODSS->getVelocity();
        mp_toSpace.plane.pos = m_MyAirPlaneTMO_ODSS->getPosition();
        mp_toSpace.plane.type = AIRPLANE_TYPE_MYAIRPLANE;
        mp_toSpace.Type =
MSG_TYPE_FROM_MYAIRPLANE_TO_SPACE_PUSH_MYAIRPLANEINFO;


m_gate_MyAirPlaneTMO_UpdateState_SpM_to_SpaceTMO_ReceiveFromMyAirPlan
e_SvM->OnewaySR(&mp_toSpace, sizeof(mp_toSpace));

        throtle = m_MyAirPlaneTMO_ODSS->getEngineThrotle();
        flap = m_MyAirPlaneTMO_ODSS->getFlaps();
        rudder = m_MyAirPlaneTMO_ODSS->getRudder();
        spoiler = m_MyAirPlaneTMO_ODSS->getSpoilers();
        landing_gear = m_MyAirPlaneTMO_ODSS->getLandingGear();

        sprintf(buf, "1e%d#%d#%d#%d#%d#%d#%d#%d#%d", throtle.left,
throtle.right, flap.front_left, flap.front_right, flap.rear_left, flap.rear_right, rudder,
spoiler.left, spoiler.right, landing_gear);
        if(sendto(sock, buf, strlen(buf), 0, (struct sockaddr *) &server,
server_len) < 0) {
                fprintf(stderr, "Error at client sendto for etc information
");
                closesocket(sock);
                exit(5);
        }

        sprintf(buf, "current enginethrotle : %d, %d₩n current flaps : %d,
%d, %d, %d₩n current rudder :%d₩n", throtle.left, throtle.right, flap.front_left,
flap.front_right, flap.rear_left, flap.rear_right, rudder);
        TMOSLprintf(buf);

        pos = m_MyAirPlaneTMO_ODSS->getPosition();
        sprintf(buf, "1p%d#%d#%d", (int) (pos.x), (int) (pos.y), (int) (pos.z));

        if(sendto(sock, buf, strlen(buf), 0, (struct sockaddr *) &server,
server_len) < 0) {
                fprintf(stderr, "Error at client sendto ");
                closesocket(sock);
                exit(5);
        }
        vel = m_MyAirPlaneTMO_ODSS->getVelocity();
        sprintf(buf, "1v%d#%d#%d", (int) (vel.x), (int) (vel.y), (int) (vel.z));
```

```
		if(sendto(sock, buf, strlen(buf), 0, (struct sockaddr *) &server,
server_len) < 0) {
			fprintf(stderr, "Error at client sendto ");
			closesocket(sock);
			exit(5);
		}
}

MyAirPlaneTMO_UpdateState_SpM::MyAirPlaneTMO_UpdateState_SpM(const char
* SpM_name, TMOGateClass &gate1, TMOGateClass & gate2,
MyAirPlaneTMO_ODSS &odss, access_mode_type mode)
{
		m_MyAirPlaneTMO_ODSS = &odss;

m_gate_MyAirPlaneTMO_UpdateState_SpM_to_SpaceTMO_ReceiveFromMyAirPlan
e_SvM = &gate1;

m_gate_MyAirPlaneTMO_UpdateState_SpM_to_ControlTowerTMO_ReceiveFromMy
AirPlane_SvM = &gate2;
		build_regist_info_ODSS(m_MyAirPlaneTMO_ODSS->get_id(), mode);
		build_regist_info_SpM_name(SpM_name);

		MicroSec	from = 5;
		from *= 1000 * 1000;					// Start
after 5 seconds when process began
		MicroSec	until = 2 * 60 * 60;
		until *= 1000 * 1000;					// Finish 2
hours later
		MicroSec every = TIME_UNIT * 1000;			// SpM1
must run every TIME_UNIT ms
		MicroSec est = 0 * 1000;

		MicroSec lst = 5 * 1000;
		MicroSec by = 10 * 1000;

		AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
		build_regist_info_AAC(AAC1);

		if(RegisterSpM() == FAIL)
			TMOSLprintf("Fail to register
MyAirPlaneTMO_UpdateState_SpM object ₩n");
		else
			TMOSLprintf("Succeed to register
MyAirPlaneTMO_UpdateState_SpM object ₩n");
}

MyAirPlaneTMO_UpdateState_SpM::~MyAirPlaneTMO_UpdateState_SpM()
{
		;
}



//
//
//		Filename	:		SpaceTMO.h & SpaceTMO.cpp
//		Description :		Class def. of SpaceTMO and its
implementation
//		Author		:		Min-Gu Lee
//
//

#ifndef SpaceTMO_
#define SpaceTMO_

#include "TMOSLv2.h"
#include "SpaceTMO_ODSS.h"
#include "SpaceTMO_UpdateState_SpM.h"
#include "SpaceTMO_UpdateWeatherOthers_SpM.h"
#include "SpaceTMO_ReceiveFromMyAirPlane_SvM.h"
#include "SpaceTMO_ReceiveFromOutSpace_SvM.h"

using namespace TMO;

class SpaceTMO : public TMOBaseClass
{
```

```
private :
        SpaceTMO_ODSS       m_SpaceTMO_ODSS;
        SpaceTMO_UpdateState_SpM       m_SpaceTMO_UpdateState_SpM;
        SpaceTMO_UpdateWeatherOthers_SpM
m_SpaceTMO_UpdateWeatherOthers_SpM;
        SpaceTMO_ReceiveFromMyAirPlane_SvM
m_SpaceTMO_ReceiveFromMyAirPlane_SvM;
        SpaceTMO_ReceiveFromOutSpace_SvM
m_SpaceTMO_ReceiveFromOutSpace_SvM;
public :
        SpaceTMO(const char * TMO_name, TMOGateClass & gate1,
TMOGateClass & gate2, TMOGateClass & gate3, tms & TMO_start_time);
};

#endif


#include "SpaceTMO.h"

SpaceTMO::SpaceTMO(const char * TMO_name, TMOGateClass & gate1,
TMOGateClass & gate2, TMOGateClass & gate3, tms & TMO_start_time)
:
        m_SpaceTMO_UpdateState_SpM("SpaceTMO_UpdateState_SpM",
gate1, gate2, gate3, m_SpaceTMO_ODSS, RW),

m_SpaceTMO_UpdateWeatherOthers_SpM("SpaceTMO_UpdateWeatherOthers_SpM
", gate1, gate2, gate3, m_SpaceTMO_ODSS, RW),

m_SpaceTMO_ReceiveFromMyAirPlane_SvM("SpaceTMO_ReceiveFromMyAirPlane
_SvM", m_SpaceTMO_ODSS, RW),

m_SpaceTMO_ReceiveFromOutSpace_SvM("SpaceTMO_ReceiveFromOutSpace_Sv
M", m_SpaceTMO_ODSS, RW)
{
        activate(TMO_name, TMO_start_time);
};
```

```
//
//
//      Filename    :       SpaceTMO_ODSS.h & SpaceTMO_ODSS.cpp
//      Description :       Class def. of SpaceTMO_ODSS and its
implementation
//      Author      :       Min-Gu Lee
//
//


#ifndef SpaceTMO_ODSS_
#define     SpaceTMO_ODSS_

#include "TMOSLv2.h"
#include "inc.h"


using namespace TMO;


class SpaceTMO_ODSS : public ODSSBaseClass
{
private :
        struct      weather     m_weather;
        struct AirPlane         m_MyAirPlaneInfo;
        struct AirPlane         m_AirPlaneInfo[MAX_AIRPLANES];
        int         m_Runway;

public :
        void updateAirPlanes();
        struct AirPlane getMyAirPlaneInfo();
        void setWeather(struct weather w);
        struct weather getWeather();
        SpaceTMO_ODSS();
        int insertAirPlane(struct AirPlane plane);
        int removeAirPlane(int seq);
        struct AirPlane getAirPlane(int seq);
        void setMyAirPlaneInfo(struct AirPlane myairplaneinfo);
```

```
};

#endif



#include "SpaceTMO_ODSS.h"

SpaceTMO_ODSS::SpaceTMO_ODSS()
{
        m_Runway = RUNWAY_AVAILABLE;
        for(int i = 0; i < MAX_AIRPLANES; i++) {
                m_AirPlaneInfo[i].id = -1;
        }
        m_weather.airpressure = 1000;
        m_weather.wind.x = 0;
        m_weather.wind.y = 0;
        m_weather.wind.z = 0;

};

struct AirPlane
SpaceTMO_ODSS::getAirPlane(int seq)
{
        return m_AirPlaneInfo[seq];
};

int
SpaceTMO_ODSS::insertAirPlane(AirPlane plane)
{
        for(int i = 0; i < MAX_AIRPLANES; i++) {
                if(m_AirPlaneInfo[i].id == -1) {
                        EnterODSS_RW();
                        m_AirPlaneInfo[i] = plane;
                        ExitODSS_RW();
                        return 1;
                }
        }
        return -1;
}

int
SpaceTMO_ODSS::removeAirPlane(int seq)
{
        if(m_AirPlaneInfo[seq].id != -1) {
                EnterODSS_RW();
                m_AirPlaneInfo[seq].id = -1;
                ExitODSS_RW();
                return 1;
        } else
                return -1;
}


struct weather
SpaceTMO_ODSS::getWeather()
{
        return m_weather;
}

void
SpaceTMO_ODSS::setWeather(weather w)
{
        EnterODSS_RW();
        m_weather = w;
        ExitODSS_RW();
}

void
SpaceTMO_ODSS::setMyAirPlaneInfo(struct AirPlane myairplaneinfo)
{
        EnterODSS_RW();
        m_MyAirPlaneInfo = myairplaneinfo;
        ExitODSS_RW();
}


struct AirPlane
```

```
SpaceTMO_ODSS::getMyAirPlaneInfo()
{
        return m_MyAirPlaneInfo;
}

void
SpaceTMO_ODSS::updateAirPlanes()
{
// the other airplanes are moving randomly
        for(int i = 0; i < MAX_AIRPLANES; i++) {
                if(m_AirPlaneInfo[i].id != -1) {
                        EnterODSS_RW();
                        m_AirPlaneInfo[i].vel.x += (rand() % 100 -
50) * TIME_UNIT / 1000;
                        m_AirPlaneInfo[i].vel.y += (rand() % 100 -
50) * TIME_UNIT / 1000;
                        m_AirPlaneInfo[i].vel.z += (rand() % 100 -
50) * TIME_UNIT / 1000;
                        m_AirPlaneInfo[i].pos.x =
m_AirPlaneInfo[i].pos.x + m_AirPlaneInfo[i].vel.x * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                        m_AirPlaneInfo[i].pos.y =
m_AirPlaneInfo[i].pos.y + m_AirPlaneInfo[i].vel.y * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                        m_AirPlaneInfo[i].pos.z =
m_AirPlaneInfo[i].pos.z + m_AirPlaneInfo[i].vel.z * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                        ExitODSS_RW();
                }
        }
}


//
//
//      Filename    :       SpaceTMO_ReceiveFromMyAirPlane_SvM.h &
SpaceTMO_ReceiveFromMyAirPlane_SvM.cpp
//      Description :       Class def. of
SpaceTMO_ReceiveFromMyAirPlane_SvM and its implementation
//      Author      :       Min-Gu Lee
//
//

#ifndef SpaceTMO_ReceiveFromMyAirPlane_SvM_
#define     SpaceTMO_ReceiveFromMyAirPlane_SvM_

#include "TMOSLv2.h"
#include "inc.h"
#include "SpaceTMO_ODSS.h"

using namespace TMO;

class SpaceTMO_ReceiveFromMyAirPlane_SvM : public SvMBaseClass
{
private :
        ParamStruct_FromMyAirPlane_to_Space     mp_fromMyAirPlane;
        SpaceTMO_ODSS       *       m_SpaceTMO_ODSS;
public :
        SpaceTMO_ReceiveFromMyAirPlane_SvM(const char *,
SpaceTMO_ODSS &, access_mode_type);
        void SvMBody();
};

#endif


#include "SpaceTMO_ReceiveFromMyAirPlane_SvM.h"

SpaceTMO_ReceiveFromMyAirPlane_SvM::SpaceTMO_ReceiveFromMyAirPlane_Sv
M(const char * SvM_name, SpaceTMO_ODSS &odss, access_mode_type mode)
{
        build_regist_info_SvM_name(SvM_name);
        m_SpaceTMO_ODSS = &odss;
        build_regist_info_ODSS(m_SpaceTMO_ODSS->get_id(), mode);
        build_regist_info_guranteed_completion_time(20 * 2000);
        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
SpaceTMO_ReceiveFromMyAirPlane_SvM₩n");
```

```
		} else {
			TMOSLprintf("Succeed to register
SpaceTMO_ReceiveFromMyAirPlane_SvM₩n");
		}
}

void
SpaceTMO_ReceiveFromMyAirPlane_SvM::SvMBody()
{
		int		Client_RRQID;
		tmsp		timestamp;
		char buf[256];

		ReceiveSR(Client_RRQID, &mp_fromMyAirPlane,
sizeof(mp_fromMyAirPlane), timestamp);
		if(mp_fromMyAirPlane.Type ==
MSG_TYPE_FROM_MYAIRPLANE_TO_SPACE_PUSH_MYAIRPLANEINFO) {

m_SpaceTMO_ODSS->setMyAirPlaneInfo(mp_fromMyAirPlane.plane);
		}
		sprintf(buf, "Current Pos : %f, %f, %f₩n",
mp_fromMyAirPlane.plane.pos.x, mp_fromMyAirPlane.plane.pos.y,
mp_fromMyAirPlane.plane.pos.z);
		TMOSLprintf(buf);
}



//
//
//		Filename	:		SpaceTMO_ReceiveFromOutSpace_SvM.h &
SpaceTMO_ReceiveFromOutSpace_SvM.cpp
//		Description :		Class def. of
SpaceTMO_ReceiveFromOutSpace_SvM and its implementation
//		Author		:		Min-Gu Lee
//
//


#ifndef SpaceTMO_ReceiveFromOutSpace_SvM_
#define SpaceTMO_ReceiveFromOutSpace_SvM_


#include "TMOSLv2.h"
#include "inc.h"
#include "SpaceTMO_ODSS.h"

class SpaceTMO_ReceiveFromOutSpace_SvM : public SvMBaseClass
{
private :
		SpaceTMO_ODSS	*		m_SpaceTMO_ODSS;
		struct ParamStruct_FromOutSpace_to_Space mp_fromOutSpace;
public :
		void SvMBody();
		SpaceTMO_ReceiveFromOutSpace_SvM(const char * SvM_name,
SpaceTMO_ODSS & odss, access_mode_type mode);
};

#endif
#include "SpaceTMO_ReceiveFromOutSpace_SvM.h"


SpaceTMO_ReceiveFromOutSpace_SvM::SpaceTMO_ReceiveFromOutSpace_SvM(c
onst char *SvM_name, SpaceTMO_ODSS &odss, access_mode_type mode)
{
		build_regist_info_SvM_name(SvM_name);
		m_SpaceTMO_ODSS = &odss;
		build_regist_info_ODSS(m_SpaceTMO_ODSS->get_id(), mode);
		build_regist_info_guranteed_completion_time(20 * 2000);
		if(RegisterSvM() == FAIL) {
			TMOSLprintf("Fail to register
SpaceTMO_ReceiveFromOutSpace_SvM₩n");
		} else {
			TMOSLprintf("Succeed to register
SpaceTMO_ReceiveFromOutSpace_SvM₩n");
		}
}

void SpaceTMO_ReceiveFromOutSpace_SvM::SvMBody()
```

```
{
        int         Client_RRQID;
        tmsp        timestamp;
        char buf[256];

        mp_fromOutSpace.Type = -1;
        ReceiveSR(Client_RRQID, &mp_fromOutSpace,
sizeof(mp_fromOutSpace), timestamp);
        if(mp_fromOutSpace.Type ==
MSG_TYPE_FROM_OUTSPACE_TO_SPACE_PUSH_NEW_AIRPLANE) {
                if(mp_fromOutSpace.plane.id != -1)

m_SpaceTMO_ODSS->insertAirPlane(mp_fromOutSpace.plane);
        }
        sprintf(buf, "Receive AirPlane from OutSpace : pos - %f, %f, %f and
vel = %f, %f, %f₩n", mp_fromOutSpace.plane.pos.x,
mp_fromOutSpace.plane.pos.y, mp_fromOutSpace.plane.pos.z,
mp_fromOutSpace.plane.vel.x, mp_fromOutSpace.plane.vel.y,
mp_fromOutSpace.plane.vel.z);
        TMOSLprintf(buf);
}


//
//
//      Filename    :       SpaceTMO_UpdateState_SpM.h &
SpaceTMO_UpdateState_SpM.cpp
//      Description :       Class def. of SpaceTMO_UpdateState_SpM
and its implementation
//      Author      :       Min-Gu Lee
//
//

#ifndef SpaceTMO_UpdateState_SpM_
#define SpaceTMO_UpdateState_SpM_

#include "TMOSLv2.h"
#include "SpaceTMO_ODSS.h"
#include "inc.h"
```

```
#include <time.h>
#include <stdlib.h>

using namespace TMO;


extern int  sock, n, server_len;
extern struct       sockaddr_in         server, client;
extern struct hostent *host;


class SpaceTMO_UpdateState_SpM : public SpMBaseClass
{
private :
        SpaceTMO_ODSS       *       m_SpaceTMO_ODSS;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_MyAirPlaneTMO_ReceiveFromSpace_Sv
M;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_ControlTower_ReceiveFromSpace_SvM;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_OutSpaceTMO_ReceiveAirPlaneFromSpac
e_SvM;
        struct ParamStruct_FromSpace_to_MyAirPlane mp_toMyAirPlane;
        struct ParamStruct_FromSpace_to_ControlTower
mp_toControlTower;
        struct ParamStruct_FromSpace_to_OutSpace    mp_toOutSpace;
public :
        int m_flag;
        SpaceTMO_UpdateState_SpM(const char * SpM_name, TMOGateClass
& gate1, TMOGateClass &gate2, TMOGateClass &gate3, SpaceTMO_ODSS &
odss, access_mode_type mode);
        virtual void SpMBody();
};
#endif


#include "SpaceTMO_UpdateState_SpM.h"
```

```
SpaceTMO_UpdateState_SpM::SpaceTMO_UpdateState_SpM(const char
*SpM_name,TMOGateClass &gate1, TMOGateClass &gate2, TMOGateClass
&gate3, SpaceTMO_ODSS &odss, access_mode_type mode)
{
        m_SpaceTMO_ODSS = &odss;

m_gate_SpaceTMO_UpdateState_SpM_to_MyAirPlaneTMO_ReceiveFromSpace_Sv
M = &gate1;

m_gate_SpaceTMO_UpdateState_SpM_to_ControlTower_ReceiveFromSpace_SvM =
&gate2;

m_gate_SpaceTMO_UpdateState_SpM_to_OutSpaceTMO_ReceiveAirPlaneFromSpac
e_SvM = &gate3;
        build_regist_info_ODSS(m_SpaceTMO_ODSS->get_id(), mode);
        build_regist_info_SpM_name(SpM_name);

        MicroSec   from = 5;
        from *= 1000 * 1000;                                    // Start
after 5 seconds when process began
        MicroSec   until = 2 * 60 * 60;
        until *= 1000 * 1000;                                   // Finish 2
hours later
        MicroSec every = TIME_UNIT * 1000;                      // SpM1
must run every TIME_UNIT ms
        MicroSec est = 0 * 1000;

        MicroSec lst = 5 * 1000;
        MicroSec by = 10 * 1000;

        AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
        build_regist_info_AAC(AAC1);

        if(RegisterSpM() == FAIL)
                TMOSLprintf("Fail to register
SpaceTMO_UpdateState_SpM object ₩n");
        else
                TMOSLprintf("Succeed to register
SpaceTMO_UpdateState_SpM object ₩n");

        m_flag = 1;
}

void
SpaceTMO_UpdateState_SpM::SpMBody()
{
        srand( (unsigned)time( NULL ) );
        char buf[256];
        m_flag *= -1;
        AirPlane    plane;
        AirPlane    MyAirPlane;
        MyAirPlane = m_SpaceTMO_ODSS->getMyAirPlaneInfo();

        for(int i = 0; i < MAX_AIRPLANES; i++) {
                plane = m_SpaceTMO_ODSS->getAirPlane(i);
                if (plane.id != -1) {
                        if(plane.pos.x < 0 || plane.pos.x >
SPACE_X_MAX || plane.pos.y < 0 || plane.pos.y > SPACE_Y_MAX ||
plane.pos.z < 0 || plane.pos.z > SPACE_Z_MAX) {
                                mp_toOutSpace.plane = plane;
                                mp_toOutSpace.Type =
MSG_TYPE_FROM_SPACE_TO_OUTSPACE_PUSH_OLD_AIRPLANE;

m_gate_SpaceTMO_UpdateState_SpM_to_OutSpaceTMO_ReceiveAirPlaneFromSpac
e_SvM->OnewaySR(&mp_toOutSpace, sizeof(mp_toOutSpace));

m_SpaceTMO_ODSS->removeAirPlane(i);
                        } else {
                                mp_toControlTower.plane = plane;
                                mp_toControlTower.Type =
MSG_TYPE_FROM_SPACE_TO_CONTROLTOWER_PUSH_AIRPLANEINFO;

m_gate_SpaceTMO_UpdateState_SpM_to_ControlTower_ReceiveFromSpace_SvM->
OnewaySR(&mp_toControlTower, sizeof(mp_toControlTower));

                                // send the other air planes
information to the graphic node
```

```
                                        sprintf(buf, "3t%d%d#%d#%d", i,
(int) (plane.pos.x), (int) (plane.pos.y), (int) (plane.pos.z));
                                        server_len = sizeof(server);
                                        if(sendto(sock, buf, strlen(buf), 0,
(struct sockaddr *) &server, server_len) < 0) {
                                                fprintf(stderr, "Error at
client sendto ");
                                                closesocket(sock);
                                                exit(5);
                                        }
                                        sprintf(buf, "3v%d%d#%d#%d", i,
(int) (plane.vel.x), (int)(plane.vel.y), (int) (plane.vel.z));
                                        if(sendto(sock, buf, strlen(buf), 0,
(struct sockaddr *) &server, server_len) < 0) {
                                                fprintf(stderr, "Error at
client sendto ");
                                                closesocket(sock);
                                                exit(5);
                                        }

                                        // check crash info for myairplane
                                        if( abs(plane.pos.x -
MyAirPlane.pos.x) < abs(plane.sz.x +  MyAirPlane.sz.x) ) {
                                                sprintf(buf, "3cx");
                                        } else if (abs(plane.pos.y -
MyAirPlane.pos.y) < abs(plane.sz.y +  MyAirPlane.sz.y) ) {
                                                sprintf(buf, "3cy");
                                        } else if (abs(plane.pos.z -
MyAirPlane.pos.z) < abs(plane.sz.z +  MyAirPlane.sz.z) ) {
                                                sprintf(buf, "3cz");
                                        } else {
                                                sprintf(buf, "0");
                                        }
                                        if(sendto(sock, buf, strlen(buf), 0,
(struct sockaddr *) &server, server_len) < 0) {
                                                fprintf(stderr, "Error at
client sendto ");
                                                closesocket(sock);
                                                exit(5);
                                        }

                                        // check landing status
                                        int state;
                                        if (MyAirPlane.pos.z == 0) {
                                                if( (MyAirPlane.pos.x >
59950) && (MyAirPlane.pos.y < 60050) && (MyAirPlane.pos.y > 49980) &&
(MyAirPlane.pos.y < 50040)) {
                                                        if(
MyAirPlane.vel.x < 100 && MyAirPlane.vel.y < 10 && MyAirPlane.vel.z < 10) {

state = 0;
                                                        } else {

state = 1; // 활주로를 속도가 너무 빨라서 벗어남
                                                        }
                                                } else {
                                                        state = 2;
// 활주로가 아닌곳에 착륙했음
                                                }
                                        //      sprintf(buf, "활주로 착륙
상태 state = %d", state);
                                        //      TMOSLprintf(buf);
                                                sprintf(buf, "3s%d",
state);
                                                if(sendto(sock, buf,
strlen(buf), 0, (struct sockaddr *) &server, server_len) < 0) {

fprintf(stderr, "Error at client sendto ");

closesocket(sock);
                                                        exit(5);
                                                }
                                        }
                                }
                        }
                }
        }
}
```

```
//
//
//          Filename    :           SpaceTMO_UpdateWeatherOthers_SpM.h &
SpaceTMO_UpdateWeatherOthers_SpM.cpp
//          Description :           Class def. of
SpaceTMO_UpdateWeatherOthers_SpM and its implementation
//          Author      :           Min-Gu Lee
//
//


#ifndef SpaceTMO_UpdateWeatherOthers_SpM_
#define SpaceTMO_UpdateWeatherOthers_SpM_

#include "TMOSLv2.h"
#include "SpaceTMO_ODSS.h"
#include "inc.h"
#include <time.h>
#include <stdlib.h>

using namespace TMO;


extern int  sock, n, server_len;
extern struct           sockaddr_in             server, client;
extern struct hostent *host;


class SpaceTMO_UpdateWeatherOthers_SpM : public SpMBaseClass
{
private :
        SpaceTMO_ODSS       *           m_SpaceTMO_ODSS;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_MyAirPlaneTMO_ReceiveFromSpace_Sv
M;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_ControlTower_ReceiveFromSpace_SvM;
        TMOGateClass        *
m_gate_SpaceTMO_UpdateState_SpM_to_OutSpaceTMO_ReceiveAirPlaneFromSpac
e_SvM;
        struct ParamStruct_FromSpace_to_MyAirPlane mp_toMyAirPlane;
        struct ParamStruct_FromSpace_to_ControlTower
mp_toControlTower;
        struct ParamStruct_FromSpace_to_OutSpace    mp_toOutSpace;
public :
        int m_flag;
        SpaceTMO_UpdateWeatherOthers_SpM(const char * SpM_name,
TMOGateClass & gate1, TMOGateClass &gate2, TMOGateClass &gate3,
SpaceTMO_ODSS & odss, access_mode_type mode);
        virtual void SpMBody();
};
#endif


#include "SpaceTMO_UpdateWeatherOthers_SpM.h"

SpaceTMO_UpdateWeatherOthers_SpM::SpaceTMO_UpdateWeatherOthers_SpM(co
nst char *SpM_name,TMOGateClass &gate1, TMOGateClass &gate2,
TMOGateClass &gate3, SpaceTMO_ODSS &odss, access_mode_type mode)
{
        m_SpaceTMO_ODSS = &odss;

m_gate_SpaceTMO_UpdateState_SpM_to_MyAirPlaneTMO_ReceiveFromSpace_Sv
M = &gate1;

m_gate_SpaceTMO_UpdateState_SpM_to_ControlTower_ReceiveFromSpace_SvM =
&gate2;

m_gate_SpaceTMO_UpdateState_SpM_to_OutSpaceTMO_ReceiveAirPlaneFromSpac
e_SvM = &gate3;
        build_regist_info_ODSS(m_SpaceTMO_ODSS->get_id(), mode);
        build_regist_info_SpM_name(SpM_name);

        MicroSec    from = 5;
```

```
	from *= 1000 * 1000;					// Start
after 5 seconds when process began
	MicroSec	until = 2 * 60 * 60;
	until *= 1000 * 1000;					// Finish 2
hours later
	MicroSec every = TIME_UNIT * 1000;		// SpM1
must run every TIME_UNIT ms
	MicroSec est = 0 * 1000;

	MicroSec lst = 5 * 1000;
	MicroSec by = 10 * 1000;

	AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
	build_regist_info_AAC(AAC1);

	if(RegisterSpM() == FAIL)
		TMOSLprintf("Fail to register
SpaceTMO_UpdateState_SpM object ₩n");
	else
		TMOSLprintf("Succeed to register
SpaceTMO_UpdateState_SpM object ₩n");

	m_flag = 1;
}

void
SpaceTMO_UpdateWeatherOthers_SpM::SpMBody()
{
	srand( (unsigned)time( NULL ) );
	char buf[256];
	m_flag *= -1;
	AirPlane	plane;
	AirPlane	MyAirPlane;
	MyAirPlane = m_SpaceTMO_ODSS->getMyAirPlaneInfo();

	struct weather ap;
	ap = m_SpaceTMO_ODSS->getWeather();
	ap.airpressure = 1000 +  m_flag * (rand() % 20);
	m_SpaceTMO_ODSS->setWeather(ap);
	mp_toMyAirPlane.w = ap;

m_gate_SpaceTMO_UpdateState_SpM_to_MyAirPlaneTMO_ReceiveFromSpace_Sv
M->OnewaySR(&mp_toMyAirPlane, sizeof(mp_toMyAirPlane));
	m_SpaceTMO_ODSS->updateAirPlanes();
}



//
//
//	Filename	:	ControlTowerTMO.h & ControlTowerTMO.cpp
//	Description :	Class def. of ControlTowerTMO and its
implementation
//	Author		:	Min-Gu Lee
//
//


#ifndef ControlTowerTMO_
#define	ControlTowerTMO_

#include "TMOSLv2.h"
#include "inc.h"
#include "ControlTowerTMO_ODSS.h"
#include "ControlTowerTMO_UpdateControlSystem_SpM.h"
#include "ControlTowerTMO_ReceiveFromSpace_SvM.h"
#include "ControlTowerTMO_ReceiveFromMyAirPlane_SvM.h"

class ControlTowerTMO		: public TMOBaseClass
{
private :
	ControlTowerTMO_ODSS		m_ControlTowerTMO_ODSS;
	ControlTowerTMO_UpdateControlSystem_SpM
m_ControlTowerTMO_UpdateControlSystem_SpM;
	ControlTowerTMO_ReceiveFromSpace_SvM
m_ControlTowerTMO_ReceiveFromSpace_SvM;
	ControlTowerTMO_ReceiveFromMyAirPlane_SvM
```

```
m_ControlTowerTMO_ReceiveFromMyAirPlane_SvM;
public :
        ControlTowerTMO(const char * TMO_name, TMOGateClass &, tms
&);
};
#endif



#include "ControlTowerTMO.h"

ControlTowerTMO::ControlTowerTMO(const char *TMO_name, TMOGateClass
&gate1, tms& TMO_start_time) :

m_ControlTowerTMO_UpdateControlSystem_SpM("ControlTowerTMO_UpdateState
_SpM", gate1, m_ControlTowerTMO_ODSS, RW),

m_ControlTowerTMO_ReceiveFromMyAirPlane_SvM("ControlTowerTMO_ReceiveF
romMyAirPlane_SvM", m_ControlTowerTMO_ODSS, RW),

m_ControlTowerTMO_ReceiveFromSpace_SvM("ControlTowerTMO_ReceiveFromS
pace_SvM", m_ControlTowerTMO_ODSS, RW)
{
        activate(TMO_name, TMO_start_time);
};



//
//
//      Filename    :       ControlTowerTMO_ODSS.h &
ControlTowerTMO_ODSS.cpp
//      Description :       Class def. of ControlTowerTMO ODSS and its
implementation
//      Author      :       Min-Gu Lee
//
//

#ifndef ControlTowerTMO_ODSS_
#define ControlTowerTMO_ODSS_

#include "TMOSLv2.h"
#include "inc.h"

using namespace TMO;

class ControlTowerTMO_ODSS : public ODSSBaseClass
{
private :
        int m_Runway_who_reserve;
        struct AirPlane m_AirPlaneInfo[MAX_AIRPLANES];
        struct AirPlane     m_MyAirPlaneInfo;
        int m_RunwayRequested;
        BOOL m_Runway_Reserved;
//      List<struct AirPlane> m_List;
//      ListItr<struct AirPlane> m_Ptr;
public :
        void setRunwayWhoReserve(int rsv);
        int getRunwayWhoReserve();
        void setRunwayReserved(BOOL rsv);
        BOOL getRunwayReserved();
        int getRunwayRequested();
        void resetRunwayRequested();
        void setRunwayRequested();
        struct AirPlane getMyAirPlaneInfo();
        ControlTowerTMO_ODSS();
        int                 CheckCrashing();
        int     insertAirPlane(struct AirPlane plane);
        int     removeAirPlane(int seq);
        struct AirPlane     getAirPlane(int seq);
        void    emptyAirPlane();
        void    setMyAirPlaneInfo(struct AirPlane myairplane);
//      List<struct AirPlane> m_List;
//      ListItr<struct AirPlane> m_Ptr;
protected:

};
```

```
#endif


#include "ControlTowerTMO_ODSS.h"

ControlTowerTMO_ODSS::ControlTowerTMO_ODSS()
{
        emptyAirPlane();
        m_RunwayRequested = 0;
}

int
ControlTowerTMO_ODSS::CheckCrashing()
{
        struct      size        sz;
        int tmpalarm = ALARM_LEVEL_0;
        int         alarm = ALARM_LEVEL_0;
        float tmp;

        for( int i = 0; i < MAX_AIRPLANES; i++ ) {
                if(m_AirPlaneInfo[i].id != -1) {
                        tmpalarm = ALARM_LEVEL_0;
                        if((tmp = sqrt(pow(m_MyAirPlaneInfo.pos.x -
m_AirPlaneInfo[i].pos.x, 2) + pow(m_MyAirPlaneInfo.pos.y -
m_AirPlaneInfo[i].pos.y, 2) + pow(m_MyAirPlaneInfo.pos.z -
m_AirPlaneInfo[i].pos.z, 2))) < MARGIN_LEVEL3) {
                                tmpalarm = ALARM_LEVEL_3;
                        } else if (tmp < MARGIN_LEVEL2) {
                                tmpalarm = ALARM_LEVEL_2;
                        } else if (tmp < MARGIN_LEVEL1) {
                                tmpalarm = ALARM_LEVEL_1;
                        } else {
                                tmpalarm = ALARM_LEVEL_0;
                        }
                        if(alarm < tmpalarm)
                                alarm = tmpalarm;
                }
        }
        return alarm;
}

struct AirPlane
ControlTowerTMO_ODSS::getAirPlane(int seq)
{
        return m_AirPlaneInfo[seq];
}

int
ControlTowerTMO_ODSS::insertAirPlane(struct AirPlane plane)
{
        for(int i = 0; i < MAX_AIRPLANES; i++ ) {
                if( m_AirPlaneInfo[i].id == -1 ) {
                        EnterODSS_RW();
                        m_AirPlaneInfo[i] = plane;
                        ExitODSS_RW();
                        return 1;
                }
        }
        return -1;
}

int
ControlTowerTMO_ODSS::removeAirPlane(int seq)
{
        if(m_AirPlaneInfo[seq].id != -1) {
                EnterODSS_RW();
                m_AirPlaneInfo[seq].id = -1;
                ExitODSS_RW();
                return 1;
        } else
                return -1;

}

void
ControlTowerTMO_ODSS::emptyAirPlane()
{
        EnterODSS_RW();
```

```
	for(int i = 0; i < MAX_AIRPLANES; i++) {
		m_AirPlaneInfo[i].id = -1;
	}
	ExitODSS_RW();
}

void
ControlTowerTMO_ODSS::setMyAirPlaneInfo(struct AirPlane myairplane)
{
	EnterODSS_RW();
	m_MyAirPlaneInfo = myairplane;
	ExitODSS_RW();
}

struct AirPlane
ControlTowerTMO_ODSS::getMyAirPlaneInfo()
{
	return m_MyAirPlaneInfo;
}

void
ControlTowerTMO_ODSS::setRunwayRequested()
{
	EnterODSS_RW();
	m_RunwayRequested = 1;
	ExitODSS_RW();
}

void
ControlTowerTMO_ODSS::resetRunwayRequested()
{
	EnterODSS_RW();
	m_RunwayRequested = 0;
	ExitODSS_RW();
}

int
ControlTowerTMO_ODSS::getRunwayRequested()
{
	return m_RunwayRequested;
}

BOOL ControlTowerTMO_ODSS::getRunwayReserved()
{
	return m_Runway_Reserved;
}

void ControlTowerTMO_ODSS::setRunwayReserved(BOOL rsv)
{
	m_Runway_Reserved = rsv;
}

int ControlTowerTMO_ODSS::getRunwayWhoReserve()
{
	return m_Runway_who_reserve;
}
void ControlTowerTMO_ODSS::setRunwayWhoReserve(BOOL rsv)
{
	m_Runway_who_reserve = rsv;
}


//
//
//	Filename	:
ControlTowerTMO_ReceiveFromMyAirPlane_SvM.h &
ControlTowerTMO_ReceiveFromMyAirPlane_SvM.cpp
//	Description :	Class def. of
ControlTowerTMO_ReceiveFromMyAirPlane_SvM and its implementation
//	Author	:	Min-Gu Lee
//
//

#ifndef ControlTowerTMO_ReceiveFromMyAirPlane_SvM_
#define ControlTowerTMO_ReceiveFromMyAirPlane_SvM_

#include "TMOSLv2.h"
#include "ControlTowerTMO_ODSS.h"
```

```
#include "inc.h"

using namespace TMO;

class ControlTowerTMO_ReceiveFromMyAirPlane_SvM : public SvMBaseClass
{
private :
        ControlTowerTMO_ODSS            *m_ControlTowerTMO_ODSS;
        struct      ParamStruct_FromControlTower_to_MyAirPlane
mp_toMyAirPlane;
public:
        ControlTowerTMO_ReceiveFromMyAirPlane_SvM(const char *
SvM_name, ControlTowerTMO_ODSS & odss, access_mode_type mode);
        struct ParamStruct_FromMyAirPlane_to_ControlTower
mp_fromMyAirPlane;
        void SvMBody();
};

#endif



#include "ControlTowerTMO_ReceiveFromMyAirPlane_SvM.h"

void
ControlTowerTMO_ReceiveFromMyAirPlane_SvM::SvMBody()
{
        int         Client_RRQID;
        tmsp        timestamp;

        ReceiveSR(Client_RRQID, &mp_fromMyAirPlane,
sizeof(mp_fromMyAirPlane), timestamp);
        if(mp_fromMyAirPlane.Type ==
MSG_TYPE_FROM_MYAIRPLANE_TO_CONTROLTOWER_REQUEST_RUNWAY) {
                m_ControlTowerTMO_ODSS->setRunwayRequested();
        }
}

ControlTowerTMO_ReceiveFromMyAirPlane_SvM::ControlTowerTMO_ReceiveFro
mMyAirPlane_SvM(const char *SvM_name, ControlTowerTMO_ODSS &odss,
access_mode_type mode)
{
        build_regist_info_SvM_name(SvM_name);
        m_ControlTowerTMO_ODSS = &odss;
        build_regist_info_ODSS (m_ControlTowerTMO_ODSS->get_id(),
mode);
        build_regist_info_guranteed_completion_time(20 * 1000);
        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
ControlTowerTMO_ReceiveFromMyAirPlane_SvM₩n");
        } else {
                TMOSLprintf("Succeed to register
ControlTowerTMO_ReceiveFromMyAirPlane_SvM₩n");
        }
}



//
//
//      Filename    :       ControlTowerTMO_ReceiveFromSpace_SvM.h
& ControlTowerTMO_ReceiveFromSpace_SvM.cpp
//      Description :       Class def. of
ControlTowerTMO_ReceiveFromSpace_SvM and its implementation
//      Author      :       Min-Gu Lee
//
//


#ifndef ControlTowerTMO_ReceiveFromSpace_SvM_
#define ControlTowerTMO_ReceiveFromSpace_SvM_

#include "TMOSLv2.h"
#include "inc.h"
#include "ControlTowerTMO_ODSS.h"

class ControlTowerTMO_ReceiveFromSpace_SvM : public SvMBaseClass
{
```

```
private :
        ControlTowerTMO_ODSS            *
m_ControlTowerTMO_ODSS;
        ParamStruct_FromSpace_to_ControlTower       mp_fromSpace;
public :
        ControlTowerTMO_ReceiveFromSpace_SvM(const char *,
ControlTowerTMO_ODSS &, access_mode_type);
        void SvMBody();
};

#endif



#include "ControlTowerTMO_ReceiveFromSpace_SvM.h"

ControlTowerTMO_ReceiveFromSpace_SvM::ControlTowerTMO_ReceiveFromSpac
e_SvM(const char * SvM_name, ControlTowerTMO_ODSS & odss,
access_mode_type mode)
{
        build_regist_info_SvM_name(SvM_name);
        m_ControlTowerTMO_ODSS = &odss;
        build_regist_info_ODSS(m_ControlTowerTMO_ODSS->get_id(), mode);
        build_regist_info_guranteed_completion_time(20 * 1000);
        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
ControlTowerTMO_ReceiveFromSpace_SvM₩n");
        } else {
                TMOSLprintf("Succeed to register
ControlTowerTMO_ReceiveFromSpace_SvM₩n");
        }
}

void
ControlTowerTMO_ReceiveFromSpace_SvM::SvMBody()
{

        int         Client_RRQID;
        tmsp        timestamp;
        char buf[256];

        ReceiveSR(Client_RRQID, &mp_fromSpace, sizeof(mp_fromSpace),
timestamp);
        if (mp_fromSpace.Type ==
MSG_TYPE_FROM_SPACE_TO_CONTROLTOWER_PUSH_AIRPLANEINFO) {

m_ControlTowerTMO_ODSS->insertAirPlane(mp_fromSpace.plane);
// prints for debug
//              sprintf(buf, "Plane # %d, plane.pos : %f, %f, %f₩n",
mp_fromSpace.plane.id, mp_fromSpace.plane.pos.x, mp_fromSpace.plane.pos.y,
mp_fromSpace.plane.pos.z);
//              TMOSLprintf(buf);
        }
}




//
//
//      Filename    :
ControlTowerTMO_UpdateControlSystem_SpM.h &
ControlTowerTMO_UpdateControlSystem_SpM.cpp
//      Description :       Class def. of
ControlTowerTMO_UpdateControlSystem_SpM and its implementation
//      Author      :       Min-Gu Lee
//
//


#ifndef ControlTowerTMO_UpdateControlSystem_SpM_
#define ControlTowerTMO_UpdateControlSystem_SpM_

#include "TMOSLv2.h"
#include "ControlTowerTMO_ODSS.h"
#include "inc.h"
```

```
using namespace TMO;

class ControlTowerTMO_UpdateControlSystem_SpM : public SpMBaseClass
{
private :
        TMOGateClass            *
m_gate_ControlTowerTMO_UpdateControlSystem_SpM_to_MyAirPlaneTMO_Receiv
eFromControlTower_SvM;
        ControlTowerTMO_ODSS            * m_ControlTowerTMO_ODSS;
        struct      ParamStruct_FromControlTower_to_MyAirPlane
mp_toMyAirPlane;

public :
        ControlTowerTMO_UpdateControlSystem_SpM(const char *,
TMOGateClass &, ControlTowerTMO_ODSS &, access_mode_type);
        ~ControlTowerTMO_UpdateControlSysteM_SpM() { ; }
        virtual void SpMBody();
};

#endif



#include "ControlTowerTMO_UpdateControlSystem_SpM.h"

ControlTowerTMO_UpdateControlSystem_SpM::ControlTowerTMO_UpdateControl
System_SpM(const char * SpM_name, TMOGateClass &gate,
ControlTowerTMO_ODSS & odss, access_mode_type mode)
{

m_gate_ControlTowerTMO_UpdateControlSystem_SpM_to_MyAirPlaneTMO_Receiv
eFromControlTower_SvM = &gate;
        m_ControlTowerTMO_ODSS = & odss;
        build_regist_info_ODSS(m_ControlTowerTMO_ODSS->get_id(), mode);
        build_regist_info_SpM_name(SpM_name);
        MicroSec    from = 5;
        from *= 1000 * 1000;                                    // Start
after 5 seconds when process began
        MicroSec    until = 2 * 60 * 60;
        until *= 1000 * 1000;                                   // Finish 2
hours later
        MicroSec every = TIME_UNIT * 1000;                      // SpM1
must run every TIME_UNIT ms
        MicroSec est = 0 * 1000;

        MicroSec lst = 5 * 1000;
        MicroSec by = 10 * 1000;

        AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
        build_regist_info_AAC(AAC1);

        if(RegisterSpM() == FAIL)
                TMOSLprintf("Fail to register
ControlTowerTMO_UpdateControlSystem_SpM object ₩n");
        else
                TMOSLprintf("Succeed to register
ControlTowerTMO_UpdateControlSystem_SpM object ₩n");
}

void
ControlTowerTMO_UpdateControlSystem_SpM::SpMBody()
{
        int         alarm;
        if( m_ControlTowerTMO_ODSS->getRunwayRequested()) {
                mp_toMyAirPlane.Type =
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_REPLY_RUNWAY;

if(!m_ControlTowerTMO_ODSS->getRunwayReserved()/*m_Runway_Reserved*/) {
                        mp_toMyAirPlane.Runway = 1;

m_ControlTowerTMO_ODSS->setRunwayReserved(TRUE);/*m_Runway_Reserved
= TRUE;*/

m_ControlTowerTMO_ODSS->setRunwayWhoReserve(-1);/*m_Runway_who_reser
v = -1;*/
                } else {
```

```
if(m_ControlTowerTMO_ODSS->getRunwayWhoReserve()/*m_Runway_who_reserv
e*/ != -1) {
                                        mp_toMyAirPlane.Runway = -1;
                                } else {
                                        mp_toMyAirPlane.Runway = 1;
                                }
                        }
                        m_ControlTowerTMO_ODSS->resetRunwayRequested();

m_gate_ControlTowerTMO_UpdateControlSystem_SpM_to_MyAirPlaneTMO_Receiv
eFromControlTower_SvM->OnewaySR(&mp_toMyAirPlane,
sizeof(mp_toMyAirPlane));
                }
                alarm = m_ControlTowerTMO_ODSS->CheckCrashing();
                //TMOSLprintf("ALARM LEVEL IS %dWn", alarm);
                mp_toMyAirPlane.Alarm = alarm;
                mp_toMyAirPlane.Type =
MSG_TYPE_FROM_CONTROLTOWER_TO_MYAIRPLANE_PUSH_ALARM;

m_gate_ControlTowerTMO_UpdateControlSystem_SpM_to_MyAirPlaneTMO_Receiv
eFromControlTower_SvM->OnewaySR(&mp_toMyAirPlane,
sizeof(mp_toMyAirPlane));
                m_ControlTowerTMO_ODSS->emptyAirPlane();
}


//
//
//              Filename    :           OutSpaceTMO.h & OutSpaceTMO.cpp
//              Description :           Class def. of OutSpaceTMO and its
implementation
//              Author      :           Min-Gu Lee
//
//

#ifndef OutSpaceTMO_
#define OutSpaceTMO_

#include "TMOSLv2.h"
```

```
#include "inc.h"
#include "OutSpaceTMO_ODSS.h"
#include "OutSpaceTMO_UpdateNewAirPlane_SpM.h"
#include "OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM.h"


using namespace TMO;

class OutSpaceTMO : public TMOBaseClass
{
private :
        OutSpaceTMO_ODSS    m_OutSpaceTMO_ODSS;
        OutSpaceTMO_UpdateNewAirPlane_SpM
m_OutSpaceTMO_UpdateNewAirPlane_SpM;
        OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM
m_OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM;
public :
        OutSpaceTMO(const char * TMO_name, TMOGateClass &gate, tms
& TMO_start_time);
};

#endif


#include "OutSpaceTMO.h"


OutSpaceTMO::OutSpaceTMO(const char *TMO_name, TMOGateClass &gate,
tms &TMO_start_time) :

m_OutSpaceTMO_UpdateNewAirPlane_SpM("OutSpaceTMO_UpdateNewAirPlane_S
pM", gate, m_OutSpaceTMO_ODSS, RW),

m_OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM("OutSpaceTMO_ReceiveAirPlan
eFromSpace_SvM", m_OutSpaceTMO_ODSS, RW)
{
        activate (TMO_name, TMO_start_time);
}
```

```
//
//
//          Filename    :           OutSpaceTMO_ODSS.h &
OutSpaceTMO_ODSS.cpp
//          Description :           Class def. of OutSpaceTMO_ODSS and its
implementation
//          Author      :           Min-Gu Lee
//
//


#ifndef OutSpaceTMO_ODSS_
#define OutSpaceTMO_ODSS_

#include "TMOSLv2.h"
#include "inc.h"
#include <time.h>
#include <stdlib.h>


using namespace TMO;

class OutSpaceTMO_ODSS : public ODSSBaseClass
{
private :
        AirPlane    m_NewAirPlane;
public :
        int getNumOfGeneratedAirPlane();
        void resetGeneratedNewAirPlane();
        struct AirPlane getGeneratedNewAirPlane();
        int m_NumOfGeneratedAirPlane;
        void generateNewAirPlane();
        OutSpaceTMO_ODSS();
};

#endif


#include "SpaceTMO_ODSS.h"

SpaceTMO_ODSS::SpaceTMO_ODSS()
{
        m_Runway = RUNWAY_AVAILABLE;
        for(int i = 0; i < MAX_AIRPLANES; i++) {
                m_AirPlaneInfo[i].id = -1;
        }
        m_weather.airpressure = 1000;
        m_weather.wind.x = 0;
        m_weather.wind.y = 0;
        m_weather.wind.z = 0;

};

struct AirPlane
SpaceTMO_ODSS::getAirPlane(int seq)
{
        return m_AirPlaneInfo[seq];
};

int
SpaceTMO_ODSS::insertAirPlane(AirPlane plane)
{
        for(int i = 0; i < MAX_AIRPLANES; i++) {
                if(m_AirPlaneInfo[i].id == -1) {
                        EnterODSS_RW();
                        m_AirPlaneInfo[i] = plane;
                        ExitODSS_RW();
                        return 1;
                }
        }
        return -1;
}

int
SpaceTMO_ODSS::removeAirPlane(int seq)
{
        if(m_AirPlaneInfo[seq].id != -1) {
                EnterODSS_RW();
```

```
                        m_AirPlaneInfo[seq].id = -1;
                        ExitODSS_RW();
                        return 1;
                } else
                        return -1;
}


struct weather
SpaceTMO_ODSS::getWeather()
{
                return m_weather;
}

void
SpaceTMO_ODSS::setWeather(weather w)
{
                EnterODSS_RW();
                m_weather = w;
                ExitODSS_RW();
}

void
SpaceTMO_ODSS::setMyAirPlaneInfo(struct AirPlane myairplaneinfo)
{
                EnterODSS_RW();
                m_MyAirPlaneInfo = myairplaneinfo;
                ExitODSS_RW();
}


struct AirPlane
SpaceTMO_ODSS::getMyAirPlaneInfo()
{
                return m_MyAirPlaneInfo;
}

void
SpaceTMO_ODSS::updateAirPlanes()
{
// the other airplanes are moving randomly
                for(int i = 0; i < MAX_AIRPLANES; i++) {
                        if(m_AirPlaneInfo[i].id != -1) {
                                EnterODSS_RW();
                                m_AirPlaneInfo[i].vel.x += (rand() % 100 -
50) * TIME_UNIT / 1000;
                                m_AirPlaneInfo[i].vel.y += (rand() % 100 -
50) * TIME_UNIT / 1000;
                                m_AirPlaneInfo[i].vel.z += (rand() % 100 -
50) * TIME_UNIT / 1000;
                                m_AirPlaneInfo[i].pos.x =
m_AirPlaneInfo[i].pos.x + m_AirPlaneInfo[i].vel.x * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                                m_AirPlaneInfo[i].pos.y =
m_AirPlaneInfo[i].pos.y + m_AirPlaneInfo[i].vel.y * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                                m_AirPlaneInfo[i].pos.z =
m_AirPlaneInfo[i].pos.z + m_AirPlaneInfo[i].vel.z * TIME_UNIT / 1000 +
(rand() % 10 - 5);
                                ExitODSS_RW();
                        }
                }
}



//
//
//              Filename    :
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM.h &
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM.cpp
//              Description :           Class def. of
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM and its implementation
//              Author                  :           Min-Gu Lee
//
//

#ifndef OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM_
```

```
#define OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM_

#include "TMOSLv2.h"
#include "inc.h"
#include "OutSpaceTMO_ODSS.h"


using namespace TMO;


class OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM : public SvMBaseClass
{
private :
        OutSpaceTMO_ODSS    * m_OutSpaceTMO_ODSS;
public :
        struct ParamStruct_FromSpace_to_OutSpace mp_fromSpace;
        void SvMBody();
        OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM(const char *
SvM_name, OutSpaceTMO_ODSS & odss, access_mode_type mode);
};

#endif


#include "OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM.h"


OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM::OutSpaceTMO_ReceiveAirPlaneF
romSpace_SvM(const char * SvM_name, OutSpaceTMO_ODSS & odss,
access_mode_type mode)
{
        build_regist_info_SvM_name(SvM_name);
        m_OutSpaceTMO_ODSS = &odss;
        build_regist_info_ODSS(m_OutSpaceTMO_ODSS->get_id(), mode);
        build_regist_info_guranteed_completion_time(20 * 1000);
        if(RegisterSvM() == FAIL) {
                TMOSLprintf("Fail to register
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM₩n");
        } else {
                TMOSLprintf("Succeed to register
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM₩n");
        }
}

void
OutSpaceTMO_ReceiveAirPlaneFromSpace_SvM::SvMBody()
{
        int         Client_RRQID;
        tmsp        timestamp;

        ReceiveSR(Client_RRQID, &mp_fromSpace, sizeof(mp_fromSpace),
timestamp);
        TMOSLprintf("Removed AirPlane₩n");
}
//
//
//      Filename    :       OutSpaceTMO_UpdateNewAirPlane_SpM.h &
OutSpaceTMO_UpdateNewAirPlane_SpM.cpp
//      Description :       Class def. of
OutSpaceTMO_UpdateNewAirPlane_SpM and its implementation
//      Author              :       Min-Gu Lee
//
//


#ifndef OutSpaceTMO_UpdateNewAirPlane_SpM_
#define     OutSpaceTMO_UpdateNewAirPlane_SpM_

#include "TMOSLv2.h"
#include "inc.h"
#include <time.h>
#include <stdlib.h>
#include "OutSpaceTMO_ODSS.h"

using namespace TMO;


class OutSpaceTMO_UpdateNewAirPlane_SpM : public SpMBaseClass
```

```
{
private :
        int tmp;
        OutSpaceTMO_ODSS * m_OutSpaceTMO_ODSS;
        struct ParamStruct_FromOutSpace_to_Space    mp_toSpace;
        TMOGateClass *
m_gate_fromOutSpaceTMO_UpdateNewAirPlane_SpM_to_SpaceTMO_ReceiveFrom
OutSpace_SvM;
public :
        virtual void SpMBody();
        OutSpaceTMO_UpdateNewAirPlane_SpM(const char *SpM_name,
TMOGateClass &gate, OutSpaceTMO_ODSS &odss, access_mode_type mode);

};

#endif
#include "OutSpaceTMO_UpdateNewAirPlane_SpM.h"


OutSpaceTMO_UpdateNewAirPlane_SpM::OutSpaceTMO_UpdateNewAirPlane_SpM
(const char *SpM_name, TMOGateClass &gate, OutSpaceTMO_ODSS &odss,
access_mode_type mode)
{

m_gate_fromOutSpaceTMO_UpdateNewAirPlane_SpM_to_SpaceTMO_ReceiveFrom
OutSpace_SvM = &gate;
        m_OutSpaceTMO_ODSS = &odss;
        build_regist_info_ODSS(m_OutSpaceTMO_ODSS->get_id(), mode);
        build_regist_info_SpM_name(SpM_name);

        MicroSec    from = 5;
        from *= 1000 * 1000;                                    // Start
after 5 seconds when process began
        MicroSec    until = 2 * 60 * 60;
        until *= 1000 * 1000;                                   // Finish 2
hours later
        MicroSec every = TIME_UNIT * 1000;                      // SpM1
must run every TIME_UNIT ms
        MicroSec est = 0 * 1000;
        MicroSec lst = 5 * 1000;
        MicroSec by = 10 * 1000;

        AACclass AAC1(tm4_DCS_age(from), tm4_DCS_age(until), every, est,
lst, by, "");
        build_regist_info_AAC(AAC1);

        if(RegisterSpM() == FAIL)
                TMOSLprintf("Fail to register
OutSpaceTMO_UpdateNewAirPlane_SpM object ₩n");
        else
                TMOSLprintf("Succeed to register
OutSpaceTMO_UpdateNewAirPlane_SpM object ₩n");

        srand( (unsigned)time( NULL ) );
}

void OutSpaceTMO_UpdateNewAirPlane_SpM::SpMBody()
{
        tmp = rand() % 5;
        m_OutSpaceTMO_ODSS->generateNewAirPlane();

        if(tmp == 0) {

                mp_toSpace.plane =
m_OutSpaceTMO_ODSS->getGeneratedNewAirPlane();
                m_OutSpaceTMO_ODSS->resetGeneratedNewAirPlane();
                mp_toSpace.Type =
MSG_TYPE_FROM_OUTSPACE_TO_SPACE_PUSH_NEW_AIRPLANE;

m_gate_fromOutSpaceTMO_UpdateNewAirPlane_SpM_to_SpaceTMO_ReceiveFrom
OutSpace_SvM->OnewaySR(&mp_toSpace, sizeof(mp_toSpace));
                m_OutSpaceTMO_ODSS->resetGeneratedNewAirPlane();
        }
}
```

B. 사용자 응용 프로그램 원시 코드

```
// flsGraphic.h : main header file for the FLSGRAPHIC application
//

#if
!defined(AFX_FLSGRAPHIC_H__2455BEF2_B438_4365_9E29_BD1969F20414__INC
LUDED_)
#define
AFX_FLSGRAPHIC_H__2455BEF2_B438_4365_9E29_BD1969F20414__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif


#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicApp:
// See flsGraphic.cpp for the implementation of this class
//

class CFlsGraphicApp : public CWinApp
{
public:
        CFlsGraphicApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFlsGraphicApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation
        //{{AFX_MSG(CFlsGraphicApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member
functions here.
                //    DO NOT EDIT what you see in these blocks of
generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_FLSGRAPHIC_H__2455BEF2_B438_4365_9E29_BD1969F20414__INC
LUDED_)



// flsGraphic.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "flsGraphic.h"
```

```
#include "MainFrm.h"
#include "flsGraphicDoc.h"
#include "flsGraphicView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicApp

BEGIN_MESSAGE_MAP(CFlsGraphicApp, CWinApp)
	//{{AFX_MSG_MAP(CFlsGraphicApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping
macros here.
		//    DO NOT EDIT what you see in these blocks of
generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicApp construction

CFlsGraphicApp::CFlsGraphicApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CFlsGraphicApp object

CFlsGraphicApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicApp initialization

BOOL CFlsGraphicApp::InitInstance()
{
	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using
MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif

	// Change the registry key under which our settings are stored.
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including
MRU)

	// Register the application's document templates.  Document
templates
```

```
	//  serve as the connection between documents, frame windows and
views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CFlsGraphicDoc),
		RUNTIME_CLASS(CMainFrame),	// main SDI frame
window
		RUNTIME_CLASS(CFlsGraphicView));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// The one and only window has been initialized, so show and
update it.
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CFlsGraphicApp::OnAppAbout()
```

```
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicApp message handlers




// flsGraphicDoc.h : interface of the CFlsGraphicDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FLSGRAPHICDOC_H__30CEB5D7_A421_4689_850B_CD673421BA37
__INCLUDED_)
#define
AFX_FLSGRAPHICDOC_H__30CEB5D7_A421_4689_850B_CD673421BA37__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#ifndef engineThrotle_
#define engineThrotle_

struct engineThrotle {
	int		right;
	int left;
};

#endif

#ifndef flaps_
#define flaps_

struct flaps {
	int		front_left;
	int		front_right;
	int		rear_left;
	int		rear_right;
};

#endif

#ifndef spoilers_
#define spoilers_

struct spoilers {
	int		left;
	int		right;
};

#endif

#ifndef DEFS____
#define DEFS____

struct cartesian_vector {
	int x;
	int y;
	int z;
};

#define MAX_AIRPLANES		8
#define BUFSIZE 4096
#define AF_INET 2
#define SOCK_DGRAM 2
#define TMO_MSG_FROM_MYAIRPLANE			'1'
#define TMO_MSG_FROM_CONTROLTOWER  '2'
#define TMO_MSG_FROM_SPACE			'3'
#define TMO_MSG_FROM_OUTSPACE			'4'
#define TMO_MSG_FROM_MYAIRPLANE_POSITION	'p'
```

```
#define TMO_MSG_FROM_MYAIRPLANE_VELOCITY        'v'
#define TMO_MSG_FROM_MYAIRPLANE_OUT_SPACE       'c'
#define TMO_MSG_FROM_MYAIRPLANE_ETC_INFO        'e'
#define TMO_MSG_FROM_CONTROLTOWER_WARNING   'w'
#define TMO_MSG_FROM_SPACE_THEOTHERAIRPLANE_POSITION  't'
#define TMO_MSG_FROM_SPACE_CRASH_AIRPLANES      'c'
#define TMO_MSG_FROM_SPACE_THEOTHERAIRPLANE_VELOCITY 'v'
#define TMO_MSG_FROM_SPACE_LANDING_STATE        's'



#define MYAIRPLANE_MAX_STABLE_LANDING_SPEED   -1
#define MYAIRPLANE_MAX_STABLE_Z_VELOCITY        -2
#define MYAIRPLANE_OUT_SPACE_M_X        -5
#define MYAIRPLANE_OUT_SPACE_M_Y        -6
#define MYAIRPLANE_OUT_SPACE_P_X        -7
#define      MYAIRPLANE_OUT_SPACE_P_Y   -8
#define MYAIRPLANE_OUT_SPACE_P_Z        -9
#define MYAIRPLANE_LANED      1
#define CRASHED_AIRPLANES     -999



#define VIEW_SIZE_X  800
#define VIEW_SIZE_Y  600
#endif

#ifndef VIEW___
#define VIEW___
#define VIEW_XY       0
#define VIEW_XZ  1
#endif


class CFlsGraphicDoc : public CDocument
{
protected: // create from serialization only
            CFlsGraphicDoc();
            DECLARE_DYNCREATE(CFlsGraphicDoc)

// Attributes
public:

// Operations
public:

// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CFlsGraphicDoc)
            public:
            virtual BOOL OnNewDocument();
            virtual void Serialize(CArchive& ar);
            //}}AFX_VIRTUAL

// Implementation
public:
            int * m_landing_state;

            int *m_state;
            int m_oldstate;

            void ResetCurrentState();
            int m_View;

            virtual ~CFlsGraphicDoc();
            CWinThread *m_pThread;
            //struct cartesian_vector
*m_Pos_TheOtherAirPlanes[MAX_AIRPLANES];
            struct cartesian_vector *m_Pos_TheOtherAirPlanes;
            struct cartesian_vector *m_Pos_MyAirPlane;
            struct cartesian_vector *m_Vel_MyAirPlane;
            struct cartesian_vector *m_Vel_TheOtherAirPlanes;
            struct engineThrotle * m_throtle;
            struct flaps * m_flap;
            int           * m_rudder, * m_landing_gear;
            struct spoilers           * m_spoiler;
            int *m_Warning;
#ifdef _DEBUG
```

```
		virtual void AssertValid() const;
		virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
		//{{AFX_MSG(CFlsGraphicDoc)
			// NOTE - the ClassWizard will add and remove member
functions here.
			//    DO NOT EDIT what you see in these blocks of
generated code !
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_FLSGRAPHICDOC_H__30CEB5D7_A421_4689_850B_CD673421BA37
__INCLUDED_)
```

```
// flsGraphicDoc.cpp : implementation of the CFlsGraphicDoc class
//

#include "stdafx.h"
#include "flsGraphic.h"

#include "flsGraphicDoc.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

struct cartesian_vector Pos_TheOtherAirPlanes[MAX_AIRPLANES];
struct cartesian_vector Vel_TheOtherAirPlanes[MAX_AIRPLANES];
struct cartesian_vector Pos_MyAirPlane;
struct cartesian_vector Vel_MyAirPlane;
struct	engineThrotle throtle;
struct flaps flap;
int		rudder, landing_gear;
struct spoilers		spoiler;
int Warning;
int state;
int landing_state;
UINT	ReceiveUDP(LPVOID);


/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicDoc

IMPLEMENT_DYNCREATE(CFlsGraphicDoc, CDocument)

BEGIN_MESSAGE_MAP(CFlsGraphicDoc, CDocument)
		//{{AFX_MSG_MAP(CFlsGraphicDoc)
			// NOTE - the ClassWizard will add and remove mapping
macros here.
			//    DO NOT EDIT what you see in these blocks of
generated code!
		//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicDoc construction/destruction

CFlsGraphicDoc::CFlsGraphicDoc()
{
```

```
		// TODO: add one-time construction code here

		m_Pos_MyAirPlane = &Pos_MyAirPlane;
		m_Pos_TheOtherAirPlanes = (struct cartesian_vector *)*
&Pos_TheOtherAirPlanes;
		m_Vel_MyAirPlane = &Vel_MyAirPlane;
		m_Warning = &Warning;
		m_state = &state;
		m_Vel_TheOtherAirPlanes = (struct cartesian_vector *)*
&Vel_TheOtherAirPlanes;
		m_throtle = &throtle;
		m_flap = &flap;
		m_rudder = &rudder;
		m_landing_gear = &landing_gear;
		m_spoiler =  &spoiler;
		m_landing_state = &landing_state;
		landing_state = -1;


		m_Pos_MyAirPlane->x = -1;
		m_Pos_MyAirPlane->y = -1;
		m_Pos_MyAirPlane->z = -1;
		for(int i = 0; i < MAX_AIRPLANES; i++) {
			m_Pos_TheOtherAirPlanes[i].x = -1;
			m_Pos_TheOtherAirPlanes[i].y = -1;
			m_Pos_TheOtherAirPlanes[i].z = -1;
			m_Vel_TheOtherAirPlanes[i].x = 0;
			m_Vel_TheOtherAirPlanes[i].y = 0;
			m_Vel_TheOtherAirPlanes[i].z = 0;
		}
		m_View = VIEW_XZ;
		m_Warning = 0;
		m_pThread = AfxBeginThread(ReceiveUDP, 0);
}

CFlsGraphicDoc::~CFlsGraphicDoc()
{
}
```

```
BOOL CFlsGraphicDoc::OnNewDocument()
{
		if (!CDocument::OnNewDocument())
			return FALSE;

		// TODO: add reinitialization code here
		// (SDI documents will reuse this document)

		return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicDoc serialization

void CFlsGraphicDoc::Serialize(CArchive& ar)
{
		if (ar.IsStoring())
		{
			// TODO: add storing code here
		}
		else
		{
			// TODO: add loading code here
		}
}

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicDoc diagnostics

#ifdef _DEBUG
void CFlsGraphicDoc::AssertValid() const
{
		CDocument::AssertValid();
}

void CFlsGraphicDoc::Dump(CDumpContext& dc) const
{
```

```
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicDoc commands


UINT ReceiveUDP(LPVOID)
{
	char buf[BUFSIZ], buf1[6], buf2[6], buf3[6], buf4[5], buf5[5],
buf6[5], buf7[5], buf8[5], buf9[5], buf10[5];
	int		sock, n, server_len, client_len;
	struct sockaddr_in server, client;

	if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
		AfxMessageBox("Server Socket Open Failure");
		exit(1);
	}
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = htonl(INADDR_ANY);
	server.sin_port = htons(2000);

	if(bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0) {
		AfxMessageBox("Server UDP Bind Failure");
		exit(2);
	}

	while(1) {

		client_len = sizeof(client);
		memset(buf, 0, BUFSIZ);

		if((n=recvfrom(sock, buf, BUFSIZ, 0, (struct sockaddr *)
&client, &client_len)) <0) {
			AfxMessageBox("SERVER recvfrom");
			closesocket(sock);
			exit(4);
		}

		if(buf[0] == TMO_MSG_FROM_MYAIRPLANE) {

			if(buf[1] ==
TMO_MSG_FROM_MYAIRPLANE_POSITION) {
				sscanf((char *)(buf + 2),"%[-
0-9] # %[- 0-9] # %[- 0-9]", &buf1, &buf2, &buf3);
				Pos_MyAirPlane.x = atoi(buf1);
				Pos_MyAirPlane.y = atoi(buf2);
				Pos_MyAirPlane.z = atoi(buf3);
			} else if (buf[1] ==
TMO_MSG_FROM_MYAIRPLANE_VELOCITY) {
				sscanf((char *)(buf + 2),"%[-
0-9] # %[- 0-9] # %[- 0-9]", &buf1, &buf2, &buf3);
				Vel_MyAirPlane.x = atoi(buf1);
				Vel_MyAirPlane.y = atoi(buf2);
				Vel_MyAirPlane.z = atoi(buf3);
			} else if (buf[1] ==
TMO_MSG_FROM_MYAIRPLANE_OUT_SPACE) {
				sscanf( (char *)(buf + 2), "%[-
0-9]", &buf1);
				state = atoi(buf1);
			} else if (buf[1] ==
TMO_MSG_FROM_MYAIRPLANE_ETC_INFO) {
				sscanf( (char *)(buf + 2), "%[-
0-9] # %[- 0-9] # %[- 0-9] # %[- 0-9] # %[- 0-9] # %[-
0-9] # %[- 0-9] # %[- 0-9] # %[- 0-9]", &buf1, &buf2, &buf3, &buf4,
&buf5, &buf6, &buf7, &buf8, &buf9, &buf10);
				throtle.left = atoi(buf1);
				throtle.right = atoi(buf2);
				flap.front_left = atoi(buf3);
				flap.front_right = atoi(buf4);
				flap.rear_left = atoi(buf5);
				flap.rear_right = atoi(buf6);
				rudder = atoi(buf7);
				//AfxMessageBox(buf7);
				spoiler.left = atoi(buf8);
				spoiler.right = atoi(buf9);
				landing_gear = atoi(buf10);
```

```
                        }
                    } else if (buf[0] == TMO_MSG_FROM_CONTROLTOWER)
{
                        if(buf[1] ==
TMO_MSG_FROM_CONTROLTOWER_WARNING) {
                            Warning = 1;
                        }
                    } else if (buf[0] == TMO_MSG_FROM_SPACE) {
                        sscanf((char *)(buf + 3),"%[- 0-9] # %[-
0-9] # %[- 0-9]", &buf1, &buf2, &buf3);
                        if(buf[1] ==
TMO_MSG_FROM_SPACE_THEOTHERAIRPLANE_POSITION) {
                            sprintf(buf, "%c", buf[2]);
                            Pos_TheOtherAirPlanes[atoi(buf)].x
= atoi(buf1);
                            Pos_TheOtherAirPlanes[atoi(buf)].y
= atoi(buf2);
                            Pos_TheOtherAirPlanes[atoi(buf)].z
= atoi(buf3);
                        } else if(buf[1] ==
TMO_MSG_FROM_SPACE_THEOTHERAIRPLANE_VELOCITY) {
                            sprintf(buf, "%c", buf[2]);
                            Vel_TheOtherAirPlanes[atoi(buf)].x
= atoi(buf1);
                            Vel_TheOtherAirPlanes[atoi(buf)].y
= atoi(buf2);
                            Vel_TheOtherAirPlanes[atoi(buf)].z
= atoi(buf3);
                        } else if (buf[1] ==
TMO_MSG_FROM_SPACE_CRASH_AIRPLANES) {
                            state = CRASHED_AIRPLANES;
                        } else if (buf[1] =
TMO_MSG_FROM_SPACE_LANDING_STATE) {
                            sprintf(buf, "%c", buf[2]);
                            landing_state = atoi(buf);
                        }
                    }
            }
            sprintf(buf, "", "");
            return 1;
}

void CFlsGraphicDoc::ResetCurrentState()
{
            m_Pos_MyAirPlane->x = -1;
            m_Pos_MyAirPlane->y = -1;
            m_Pos_MyAirPlane->z = -1;
            for(int i = 0; i < MAX_AIRPLANES; i++) {
                        m_Pos_TheOtherAirPlanes[i].x = -1;
                        m_Pos_TheOtherAirPlanes[i].y = -1;
                        m_Pos_TheOtherAirPlanes[i].z = -1;
                        m_Vel_TheOtherAirPlanes[i].x = 0;
                        m_Vel_TheOtherAirPlanes[i].y = 0;
                        m_Vel_TheOtherAirPlanes[i].z = 0;
            }
}




// flsGraphicView.h : interface of the CFlsGraphicView class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FLSGRAPHICVIEW_H__5D639564_36C0_4550_B067_AC5393A721D
E__INCLUDED_)
#define
AFX_FLSGRAPHICVIEW_H__5D639564_36C0_4550_B067_AC5393A721DE__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

```
#ifndef VIEW___
#define VIEW___
#define VIEW_XY       0
#define VIEW_XZ 1
#endif

class CFlsGraphicView : public CView
{
protected: // create from serialization only
            CFlsGraphicView();
            DECLARE_DYNCREATE(CFlsGraphicView)

// Attributes
public:
            CFlsGraphicDoc* GetDocument();

// Operations
public:

// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CFlsGraphicView)
            public:
            virtual void OnDraw(CDC* pDC);  // overridden to draw this view
            virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
            virtual void OnInitialUpdate();
            virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR
lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd,
UINT nID, CCreateContext* pContext = NULL);
            protected:
            virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
            virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
            virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
            //}}AFX_VIRTUAL

// Implementation
public:
            void DrawBitmap();
            virtual ~CFlsGraphicView();
#ifdef _DEBUG
            virtual void AssertValid() const;
            virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
            //{{AFX_MSG(CFlsGraphicView)
            afx_msg void OnTimer(UINT nIDEvent);
            afx_msg void OnDestroy();
            afx_msg void OnMenuXz();
            afx_msg void OnMenuXy();
            //}}AFX_MSG
            DECLARE_MESSAGE_MAP()
private:
            BOOL m_firsttime;
};

#ifndef _DEBUG  // debug version in flsGraphicView.cpp
inline CFlsGraphicDoc* CFlsGraphicView::GetDocument()
   { return (CFlsGraphicDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_FLSGRAPHICVIEW_H__5D639564_36C0_4550_B067_AC5393A721D
E__INCLUDED_)



// flsGraphicView.cpp : implementation of the CFlsGraphicView class
```

```
//

#include "stdafx.h"
#include "flsGraphic.h"

#include "flsGraphicDoc.h"
#include "flsGraphicView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView

IMPLEMENT_DYNCREATE(CFlsGraphicView, CView)

BEGIN_MESSAGE_MAP(CFlsGraphicView, CView)
        //{{AFX_MSG_MAP(CFlsGraphicView)
        ON_WM_TIMER()
        ON_WM_DESTROY()
        ON_COMMAND(ID_MENU_XZ, OnMenuXz)
        ON_COMMAND(ID_MENU_XY, OnMenuXy)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView construction/destruction

CFlsGraphicView::CFlsGraphicView()
{
        // TODO: add construction code here

}

CFlsGraphicView::~CFlsGraphicView()
{
}

BOOL CFlsGraphicView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView drawing

void CFlsGraphicView::OnDraw(CDC* pDC)
{
        CFlsGraphicDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: add draw code for native data here

        DrawBitmap();

        pDoc->ResetCurrentState();

}

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView printing

BOOL CFlsGraphicView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}
```

```
void CFlsGraphicView::OnBeginPrinting(CDC* pDC, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
		//pDC->SetBkColor(RGB(0,0,0));
}

void CFlsGraphicView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView diagnostics

#ifdef _DEBUG
void CFlsGraphicView::AssertValid() const
{
	CView::AssertValid();
}

void CFlsGraphicView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CFlsGraphicDoc* CFlsGraphicView::GetDocument() // non-debug version is
inline
{

ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFlsGraphicDoc)));
	return (CFlsGraphicDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFlsGraphicView message handlers

void CFlsGraphicView::DrawBitmap()
{
	char txtbuf[256], xybuf[32], buf[32];
	CClientDC  dc(this);


	CDC	MemDC;
	MemDC.CreateCompatibleDC(&dc);

	CRect rect;
	GetClientRect(&rect);


	CFlsGraphicDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CBitmap bm_BackGround;
	CBitmap bm_MyAirPlane;
	CBitmap bm_TheOtherAirPlane;
	CBitmap bm_ControlTower;
	CBitmap *pOldBitmap;
	CBrush brush, *pOldBrush;

	if (pDoc->m_View == VIEW_XZ) {
		brush.CreateSolidBrush(RGB(230,250,255));
		pOldBrush = (CBrush *) dc.SelectObject(&brush);
		dc.Rectangle(0, 0, 800, 600);
		dc.SelectObject(pOldBrush);
		pOldBitmap = (CBitmap *)
bm_BackGround.LoadBitmap(IDB_Ground);
		MemDC.SelectObject(&bm_BackGround);
		dc.BitBlt(0, 590, 800, 10, &MemDC, 0, 0, SRCCOPY);
		dc.Rectangle(59950  * (VIEW_SIZE_X) / 100000, 590,
59950  * (VIEW_SIZE_X) / 100000  + 20, 593);

		bm_ControlTower.LoadBitmap(IDB_ControlTower_Side);
		MemDC.SelectObject(&bm_ControlTower);
		dc.BitBlt( 450, 515, 33, 75, &MemDC, 0,0,SRCAND);
	} else {
```

```
                        //brush.CreateSolidBrush(RGB(230,250,255));
                        //pOldBrush = (CBrush *) dc.SelectObject(&brush);
                        //dc.Rectangle(0, 0, 800, 600);
//                      dc.SelectObject(pOldBrush);
                        pOldBitmap = (CBitmap *)
bm_BackGround.LoadBitmap(IDB_Background);
                        MemDC.SelectObject(&bm_BackGround);
                        //dc.BitBlt(0, 0, 800, 599, &MemDC, 0, 0, SRCCOPY);
                        dc.BitBlt(0, 0, 800, 599, &MemDC, 0, 0, SRCCOPY);
                        dc.Rectangle(59950  * (VIEW_SIZE_X) / 100000, 49980 *
VIEW_SIZE_Y / 100000, 59950  * (VIEW_SIZE_X) / 100000  + 20, 49980 *
VIEW_SIZE_Y / 100000 + 5);
                        bm_ControlTower.LoadBitmap(IDB_ControlTower_Top);
                        MemDC.SelectObject(&bm_ControlTower);
                        dc.BitBlt( 450, 300, 28, 29, &MemDC, 0,0,SRCCOPY);

                }
                //MemDC.SelectObject(pOldBitmap);


                if((pDoc->m_Pos_MyAirPlane->x >= 0 &&
pDoc->m_Pos_MyAirPlane->y >= 0 && pDoc->m_Pos_MyAirPlane->z >= 0)) {
                        if (pDoc->m_View == VIEW_XY) {
                                if(pDoc->m_Vel_MyAirPlane->x >= 0) {

bm_MyAirPlane.LoadBitmap(IDB_TOP_MyAirPlane1);
                                } else {

bm_MyAirPlane.LoadBitmap(IDB_TOP_MyAirPlane2);
                                }
                                pOldBitmap = (CBitmap
*)MemDC.SelectObject(&bm_MyAirPlane);
                                dc.BitBlt((int)(pDoc->m_Pos_MyAirPlane->x *
VIEW_SIZE_X / 100000), (int)(pDoc->m_Pos_MyAirPlane->y * (VIEW_SIZE_Y -
20) / 100000), 40, 30, &MemDC, 0, 0, SRCCOPY);

                        } else if (pDoc->m_View == VIEW_XZ) {
                                if(pDoc->m_Vel_MyAirPlane->x >= 0) {

bm_MyAirPlane.LoadBitmap(IDB_MyAirPlane);
                                } else /*if (pDoc->m_Vel_MyAirPlane->x <
0)*/ {

bm_MyAirPlane.LoadBitmap(IDB_MyAirPlane2);
                                }
                                pOldBitmap = (CBitmap
*)MemDC.SelectObject(&bm_MyAirPlane);
                                dc.BitBlt((int)(pDoc->m_Pos_MyAirPlane->x *
VIEW_SIZE_X / 100000), (int)(VIEW_SIZE_Y - 20 -
pDoc->m_Pos_MyAirPlane->z * (VIEW_SIZE_Y - 20) / 10000), 40, 20,
&MemDC, 0, 0, SRCCOPY);

                        }
                        MemDC.SelectObject(pOldBitmap);
                }



                for(int i = 0; i < MAX_AIRPLANES; i++) {
                        if((pDoc->m_Pos_TheOtherAirPlanes[i].x >= 0 &&
pDoc->m_Pos_TheOtherAirPlanes[i].y >= 0 &&
pDoc->m_Pos_TheOtherAirPlanes[i].z >= 0)) {
                                if (pDoc->m_View == VIEW_XZ) {

if(pDoc->m_Vel_TheOtherAirPlanes[i].x >= 0) {

bm_TheOtherAirPlane.LoadBitmap(IDB_TheOtherAirPlane);
                                        } else {

bm_TheOtherAirPlane.LoadBitmap(IDB_TheOtherAirPlane2);
                                        }
                                        pOldBitmap = (CBitmap
*)MemDC.SelectObject(&bm_TheOtherAirPlane);

dc.BitBlt((int)(pDoc->m_Pos_TheOtherAirPlanes[i].x * (VIEW_SIZE_X) /
100000),(int)(pDoc->m_Pos_TheOtherAirPlanes[i].y * (VIEW_SIZE_Y - 20) /
100000), 40, 20, &MemDC, 0, 0, SRCCOPY);
```

```
                                } else if (pDoc->m_View == VIEW_XY) {

if(pDoc->m_Vel_TheOtherAirPlanes[i].x >= 0) {

bm_TheOtherAirPlane.LoadBitmap(IDB_TOP_TheOtherAirPlane1);
                                        } else {

bm_TheOtherAirPlane.LoadBitmap(IDB_TOP_TheOtherAirPlane2);
                                        }
                                        pOldBitmap = (CBitmap
*)MemDC.SelectObject(&bm_TheOtherAirPlane);

dc.BitBlt((int)(pDoc->m_Pos_TheOtherAirPlanes[i].x * (VIEW_SIZE_X) /
100000),(int)(VIEW_SIZE_Y - pDoc->m_Pos_TheOtherAirPlanes[i].z *
(VIEW_SIZE_Y - 20) / 10000), 40, 30, &MemDC, 0, 0, SRCCOPY);
                                }
                                MemDC.SelectObject(pOldBitmap);
                                bm_TheOtherAirPlane.DeleteObject();
                        }
                }

                // 왼쪽의 theotherairplanes의 정보를 출력하기 위한 panel 을 그리는
부분
                brush.DeleteObject();
                brush.CreateSolidBrush(RGB(0,0,0));
                pOldBrush = (CBrush *) dc.SelectObject(&brush);
                dc.Rectangle(800, 0, 1012, 600);
                dc.SelectObject(pOldBrush);
                brush.DeleteObject();

                // 아래쪽의 myairplane의 정보를 출력하기 위한 panel 을 그리는 부분
                brush.CreateSolidBrush(RGB(230, 230, 230));
                pOldBrush = (CBrush *) dc.SelectObject(&brush);
                dc.Rectangle(0, 600, 1012, 675);
                dc.SelectObject(pOldBrush);

                // data를 표시하기 위한 white box를 출력하기 위해 pen설정
                dc.SetROP2(R2_COPYPEN);
                brush.DeleteObject();
                brush.CreateSolidBrush(RGB(255,255,255));
                pOldBrush = (CBrush *) dc.SelectObject(&brush);



                // myaipplane의 Position의 출력
                dc.Rectangle(20 + 70 * 0, 610, 20 + 70 * 3 - 10, 610 + 20);
                dc.Rectangle(20 + 70 * 0, 640, 20 + 70 * 0 + 60, 640 + 20);
                dc.Rectangle(20 + 70 * 1, 640, 20 + 70 * 1 + 60, 640 + 20);
                dc.Rectangle(20 + 70 * 2, 640, 20 + 70 * 2 + 60, 640 + 20);
                sprintf(txtbuf, "Position - X / Y / Z");
                dc.TextOut(20 + 70 * 0 + 5, 612, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Pos_MyAirPlane->x));
                dc.TextOut(20 + 70 * 0 + 5, 612 + 30, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Pos_MyAirPlane->y));
                dc.TextOut(20 + 70 * 1 + 5, 612 + 30, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Pos_MyAirPlane->z));
                dc.TextOut(20 + 70 * 2 + 5, 612 + 30, txtbuf);

                // myairplane의 velocity 출력
                dc.Rectangle(240 + 70 * 0, 610, 240 + 70 * 3 - 10, 610 + 20);
                dc.Rectangle(240 + 70 * 0, 640, 240 + 70 * 0 + 60, 640 + 20);
                dc.Rectangle(240 + 70 * 1, 640, 240 + 70 * 1 + 60, 640 + 20);
                dc.Rectangle(240 + 70 * 2, 640, 240 + 70 * 2 + 60, 640 + 20);
                sprintf(txtbuf, "Velocity - X / Y / Z");
                dc.TextOut(240 + 70 * 0 + 5, 612, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Vel_MyAirPlane->x));
                dc.TextOut(240 + 70 * 0 + 5, 612 + 30, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Vel_MyAirPlane->y));
                dc.TextOut(240 + 70 * 1 + 5, 612 + 30, txtbuf);
                sprintf(txtbuf, "%d", (int)(pDoc->m_Vel_MyAirPlane->z));
                dc.TextOut(240 + 70 * 2 + 5, 612 + 30, txtbuf);

                // myairplane의 flap에 대한 정보 출력
                dc.Rectangle(460 + 50 * 0, 610, 460 + 50 * 4 - 10, 610 + 20);
                dc.Rectangle(460 + 50 * 0, 640, 460 + 50 * 0 + 40, 640 + 20);
                dc.Rectangle(460 + 50 * 1, 640, 460 + 50 * 1 + 40, 640 + 20);
                dc.Rectangle(460 + 50 * 2, 640, 460 + 50 * 2 + 40, 640 + 20);
                dc.Rectangle(460 + 50 * 3, 640, 460 + 50 * 3 + 40, 640 + 20);
```

```
sprintf(txtbuf, "Flaps - FR / FL / RR/ RL");
dc.TextOut(460 + 50 * 0 + 5, 612, txtbuf);
sprintf(txtbuf, "%d", (int)(pDoc->m_flap->front_left));
dc.TextOut(460 + 50 * 0 + 5, 612 + 30, txtbuf);
sprintf(txtbuf, "%d", (int)(pDoc->m_flap->front_right));
dc.TextOut(460 + 50 * 1 + 5, 612 + 30, txtbuf);
sprintf(txtbuf, "%d", (int)(pDoc->m_flap->rear_left));
dc.TextOut(460 + 50 * 2 + 5, 612 + 30, txtbuf);
sprintf(txtbuf, "%d", (int)(pDoc->m_flap->rear_right));
dc.TextOut(460 + 50 * 3 + 5, 612 + 30, txtbuf);

// myairplane의 throtle값 출력
dc.Rectangle(670 + 50 * 0, 610, 670 + 50 * 2 - 10, 610 + 20);
dc.Rectangle(670 + 50 * 0, 640, 670 + 50 * 0 + 40, 640 + 20);
dc.Rectangle(670 + 50 * 1, 640, 670 + 50 * 1 + 40, 640 + 20);
sprintf(txtbuf, "Throtle L / R");
dc.TextOut(670 + 50 * 0 + 5, 612, txtbuf);
sprintf(txtbuf, "%d", pDoc->m_throtle->left);
dc.TextOut(670 + 50 * 0 + 5, 642, txtbuf);
sprintf(txtbuf, "%d", pDoc->m_throtle->right);
dc.TextOut(670 + 50 * 1 + 5, 642, txtbuf);

// myairplane의 rudder값 출력
dc.Rectangle(780, 610, 780 + 60, 630);
dc.Rectangle(780, 640, 780 + 60, 660);
sprintf(txtbuf, "Rudder");
dc.TextOut(780 + 5, 612, txtbuf);
sprintf(txtbuf, "%d", *(pDoc->m_rudder));
dc.TextOut(780 + 5, 642, txtbuf);

// myairplane의 spoiler값 출력
dc.Rectangle(860, 610, 860 + 50 * 2 - 10, 630);
dc.Rectangle(860 + 50 * 0, 640, 860 + 50 * 0 + 40, 640 + 20);
dc.Rectangle(860 + 50 * 1, 640, 860 + 50 * 1 + 40, 640 + 20);
sprintf(txtbuf, "spoiler L / R");
dc.TextOut(860 + 50 * 0 + 5, 612, txtbuf);
sprintf(txtbuf, "%d", pDoc->m_spoiler->left);
dc.TextOut(860 + 50 * 0 + 5, 642, txtbuf);
sprintf(txtbuf, "%d", pDoc->m_spoiler->right);
dc.TextOut(860 + 50 * 1 + 5, 642, txtbuf);


// 왼쪽 panel의 초기값 출력
dc.Rectangle(805, 10 , 825, 10 + 40);
dc.Rectangle(830, 10 , 885, 10 + 20);
dc.Rectangle(890, 10 , 945, 10 + 20);
dc.Rectangle(950, 10 , 1005, 10 + 20);
dc.TextOut(810, 12 + 10, "#");
dc.TextOut(835, 12, "POS X");
dc.TextOut(895, 12, "POS Y");
dc.TextOut(955, 12, "POS Z");
dc.Rectangle(830, 10 + 20 , 885, 10 + 20 + 20);
dc.Rectangle(890, 10 + 20 , 945, 10 + 20 + 20);
dc.Rectangle(950, 10 + 20 , 1005, 10 + 20 + 20);
dc.TextOut(835, 12 + 20, "VEL X");
dc.TextOut(895, 12 + 20, "VEL Y");
dc.TextOut(955, 12 + 20, "VEL Z");

for(i = 0; i < MAX_AIRPLANES; i++) {
        dc.Rectangle(805, 70 + 60 * i , 825, 70 + 60 * i + 40);
        dc.Rectangle(830, 70 + 60 * i , 885, 70 + 60 * i + 20);
        dc.Rectangle(890, 70 + 60 * i , 945, 70 + 60 * i + 20);
        dc.Rectangle(950, 70 + 60 * i , 1005, 70 + 60 * i +
20);
        dc.Rectangle(830, 70 + 60 * i + 20, 885, 70 + 60 * i +
20 + 20);
        dc.Rectangle(890, 70 + 60 * i + 20, 945, 70 + 60 * i +
20 + 20);
        dc.Rectangle(950, 70 + 60 * i + 20, 1005, 70 + 60 * i
+ 20 + 20);
        sprintf(txtbuf, "%d", i + 1);
        dc.TextOut(810, 70 + 60 * i + 2 + 10, txtbuf);
        sprintf(txtbuf, "%d",
(int)(pDoc->m_Pos_TheOtherAirPlanes[i].x));
        dc.TextOut(835, 70 + 60 * i + 2, txtbuf);
        sprintf(txtbuf, "%d",
(int)(pDoc->m_Pos_TheOtherAirPlanes[i].y));
        dc.TextOut(895, 70 + 60 * i + 2, txtbuf);
```

```
                        sprintf(txtbuf, "%d",
(int)(pDoc->m_Pos_TheOtherAirPlanes[i].z));
                        dc.TextOut(955, 70 + 60 * i + 2, txtbuf);
                        sprintf(txtbuf, "%d",
(int)(pDoc->m_Vel_TheOtherAirPlanes[i].x));
                        dc.TextOut(835, 70 + 60 * i + 2 + 20, txtbuf);
                        sprintf(txtbuf, "%d",
(int)(pDoc->m_Vel_TheOtherAirPlanes[i].y));
                        dc.TextOut(895, 70 + 60 * i + 2 + 20, txtbuf);
                        sprintf(txtbuf, "%d",
(int)(pDoc->m_Vel_TheOtherAirPlanes[i].z));
                        dc.TextOut(955, 70 + 60 * i + 2 + 20, txtbuf);
                }



                // Landing Info
                dc.Rectangle(805, 550, 1005, 580);
                if(*(pDoc->m_landing_state) != -1) {
                        switch(*(pDoc->m_landing_state)) {
                        case 0 :
                                sprintf(txtbuf, "Landing Success");
                                break;
                        case 1:
                                sprintf(txtbuf, "Over Landing Speed");
                                break;
                        case 2:
                                sprintf(txtbuf, "Landed on not runway");
                                break;
                        }
                        dc.TextOut(810, 555, txtbuf);
                }



/*
                sprintf(buf, "");
                switch (*(pDoc->m_state)) {
                        case MYAIRPLANE_MAX_STABLE_LANDING_SPEED :
                                sprintf(buf, "CRASHED - OVER MAX STABLE
LANDING SPEED");
                                break;
                        case MYAIRPLANE_MAX_STABLE_Z_VELOCITY
:
                                sprintf(buf, "CRASHED - OVER MAX STABLE
Z VELOCITY");
                                break;
                        case MYAIRPLANE_OUT_SPACE_M_X         :
                                sprintf(buf, "GONE AWAY - MINUS X
AREA");
                                break;
                        case MYAIRPLANE_OUT_SPACE_M_Y         :
                                sprintf(buf, "GONE AWAY - MINUS Y
AREA");
                                break;
                        case MYAIRPLANE_OUT_SPACE_P_X         :
                                sprintf(buf, "GONE AWAY - PLUS X AREA");
                                break;
                        case MYAIRPLANE_OUT_SPACE_P_Y         :
                                sprintf(buf, "GONE AWAY - PLUS Y AREA");
                                break;
                        case MYAIRPLANE_OUT_SPACE_P_Z         :
                                sprintf(buf, "GONE AWAY - PLUS Z AREA");
                                break;
                        case MYAIRPLANE_LANED         :
                                sprintf(buf, "SUCCESSFULLY LANDED!!");
                                break;
                        case CRASHED_AIRPLANES :
                                sprintf(buf, "CRASHED WITH
THEOTHERPLANES");
                                break;
                }
                dc.Rectangle(500 + 20, 610, 500 + 200 + 20, 610 + 20);
                sprintf(txtbuf, "Current MyAirPlane State");
                dc.Rectangle(500 + 20, 640, 500 + 200 + 20, 640 + 20);
                dc.TextOut(505 + 20, 612, txtbuf);
                dc.TextOut(505 + 20, 642, buf);
*/
```

```
	dc.SelectObject(pOldBrush);
}

void CFlsGraphicView::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	Invalidate(FALSE);
	DrawBitmap();

	CView::OnTimer(nIDEvent);
}

void CFlsGraphicView::OnDestroy()
{
	CView::OnDestroy();

	// TODO: Add your message handler code here
	KillTimer(0);
}

void CFlsGraphicView::OnInitialUpdate()
{
	CView::OnInitialUpdate();

	// TODO: Add your specialized code here and/or call the base class
	CClientDC  dc(this);
//	dc.SetBkColor(RGB(0,0,0));
	SetTimer(0, 50, NULL);
}

void CFlsGraphicView::OnMenuXz()
{
	// TODO: Add your command handler code here
	CFlsGraphicDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	pDoc->m_View = VIEW_XZ;
}

void CFlsGraphicView::OnMenuXy()
{
	// TODO: Add your command handler code here
	CFlsGraphicDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	pDoc->m_View = VIEW_XY;
}

BOOL CFlsGraphicView::Create(LPCTSTR lpszClassName, LPCTSTR
lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd,
UINT nID, CCreateContext* pContext)
{
	// TODO: Add your specialized code here and/or call the base class
	HBRUSH hBrush = (HBRUSH)::GetStockObject(BLACK_BRUSH);
	::SetClassLong(this->GetSafeHwnd(), GCL_HBRBACKGROUND,
(LONG)hBrush);

	return CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,
pParentWnd, nID, pContext);
}

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_MAINFRM_H__405859A0_3AB5_4B59_9934_9082F05D6305__INCLU
DED_)
#define
AFX_MAINFRM_H__405859A0_3AB5_4B59_9934_9082F05D6305__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
```

```
{

protected: // create from serialization only
            CMainFrame();
            DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

//          CStatusBar m_wndStatusBar;
// Operations
public:

// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CMainFrame)
            virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
            //}}AFX_VIRTUAL

// Implementation
public:

            virtual ~CMainFrame();
#ifdef _DEBUG
            virtual void AssertValid() const;
            virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
            CStatusBar   m_wndStatusBar;
            CToolBar     m_wndToolBar;

// Generated message map functions
protected:
            //{{AFX_MSG(CMainFrame)
            afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                        // NOTE - the ClassWizard will add and remove member
functions here.
                        //    DO NOT EDIT what you see in these blocks of
generated code!
            //}}AFX_MSG
            DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_MAINFRM_H__405859A0_3AB5_4B59_9934_9082F05D6305__INCLU
DED_)




// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "flsGraphic.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
            //{{AFX_MSG_MAP(CMainFrame)
```

```
		// NOTE - the ClassWizard will add and remove mapping
macros here.
		//    DO NOT EDIT what you see in these blocks of
generated code !
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,		// status line indicator
	ID_SEPARATOR,
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
WS_VISIBLE | CBRS_TOP
		| CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar₩n");
		return -1;	// fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE0("Failed to create status bar₩n");
		return -1;	// fail to create
	}

	// TODO: Delete these three lines if you don't want the toolbar to
	//  be dockable
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

//	if(!m_wndStatusBar.Create(this) ||
!m_wndStatusBar.SetIndicators(indicators, sizeof(indicators) / sizeof(UINT)))
//	{
//		TRACE0 ("Failed to create status bar₩n");
//		return -1;
//	}

	HBRUSH hBrush = (HBRUSH)::GetStockObject(BLACK_BRUSH);
	::SetClassLong(this->GetSafeHwnd(), GCL_HBRBACKGROUND,
(LONG)hBrush);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: Modify the Window class or styles here by modifying
```

```
		//  the CREATESTRUCT cs

		cs.cx = 1024;
		cs.cy = 768;
		return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
		CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
		CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

감사의 글

대학원 2년이라는 기간동안 부족한 저를 지도해 주신 이승구 교수님께 큰 감사를 드립니다. 논문 지도뿐만 아니라 연구 활동에도 크게 도움을 주신 홍성제 교수님과 김종 교수님께도 역시 큰 감사드립니다. 제가 이렇게 졸업을 할 수 있는 것은 모두 교수님들의 지도와 도움 덕분입니다.

낯설은 포항에 적응하도록 도와주고, 힘들 때마다 조언을 해주던 연구실 선배님들과, 좋은 일도 그렇지 못한 일도 함께 하던 연구실 선후배님들께 깊이 감사드립니다. 또한 2년간 같이 생활한 룸메이트들에게도 고마움을 전합니다.

언제나 저를 응원해주고, 제가 힘들 때 저에게 항상 힘이 되어준 소중한 친구들에게 지면을 통해서나마 감사합니다. 친구들아, 너희들 덕분에 이렇게 무사히 졸업할 수 있었다. 고맙다.

무엇보다도 항상 저를 믿어 주시고 지금껏 이끌어 주신 부모님께 형언할 수 없는 감사드립니다. 또한 저의 진로에 많은 조언을 해 주신 저의 형, 누나들과 친척 어른들께도 깊은 감사드립니다.

다시 한번 부모님께 깊은 감사드립니다.

이 력 서

성　　명 : 이 민 구
생년월일 : 1977년 06월 13일
출 생 지 : 경기도 성남시
주　　소 : 경기도 성남시 중원구 상대원3동 863-2호

학 력

1996.3 - 2000.2 : 한양대학교 전자,전자통신,전파공학과군 (B.S)
2000.3 - 2002.2 : 포항공과대학교 전자컴퓨터공학과 (M.S)

경 력

2000.3 - 2002.2 : 포항공과대학교 전자컴퓨터공학과 교육 및 연구 조교