객체지향방법을 응용한 도시철도 종합시뮬레이터의 설계

Design of the Multi-Discipline Simulator for the Urban Rail Transit with Object-Based Concept

정상기[1], 조홍식[2], 이성혁[3], 이안호[4], 이승재[5]

Sang-Gi Chung, Hong-Sik Cho, Sung-Hyuk Lee, An-Ho Lee, Seung-Jae Lee

Key Words: *Simulator(시뮬레이터), Power Supply(전력공급시스템), Signalling(신호시스템), Origin-Destination(수송수요), System Engineering(시스템엔지니어링)*

Abstract

Most rail system related simulators currently used are designed to simulate only one discipline system. This obviously assumes the other discipline systems are running regularly not being affected by the system being simulated. In this paper a multi discipline simulator is proposed and its design concept is presented. A multi discipline simulator is the simulator in which major subsystems with different technical discipline are simulated simultaneously. The advantage of the simulator is in that it makes it possible to analyze the systems behavior while other discipline system vary. With this we can identify the possible multi-discipline problems and even find their solutions. A proto type simulator has been developed using object oriented programming. Object concept was judged best suitable to model the various multi-discipline self-controlling railway subsystems. It was applied to the target system, which is under development by the Korea Railroad Research Institute. The test results shows it is very useful in design verification. It could also be a good tool in research and development work to improve the system.

1. 서 론

시스템 설계자는 시스템의 설계가 고객의 요구조건을 만족시키는 것 못지 않게 그 설계가 가장 경제적임을 확인하여야 한다. 도시철도시스템의 경우 이러한 작업은 용이하지 않다. 시스템의 복잡성 때문에 분석적인 방법으로 어느 것이 최적의 설계라는 입증도 어렵고 또한 공공 대중이 이용하는 시설로 안전을 고려한 안전율을 고려하다 보면 과다 설계가 되기 십다. 이러한 문제를 해결하기 위하여 흔히 컴퓨터 시뮬레이션이 수행된다. 그러나 대부분의 시뮬레이터는 어느 한 기술분야의 하부시스템에 대해서만 시뮬레이션을 수행한다. 이 경우 다른 기술분야의 하부시스템은 항상 일정하게 동작된다고 하는 가정이 내포되어 있는 것이다. 물론 실제 시스템이 운전될 때는 모든 하부시스템이 서로 영향을 주면서 운전된다. 본문에서 제시되는 종합시뮬레이터는 시스템의 주요 하부시스템이 서로 영향을 주면서 동시에 시뮬레이션이 수행되는 시뮬레이터를 의미하고 그 주요 목적은 각 하부시스템의 주요 파라메타를 변경하면서 시스템의 종합 성능을 용이하게 확인할 수 있게 하는 것이다. 그리하여 최적의 설계를 위한 설계 Tool로서 이용되고 또한 고객에 설계시스템의 성능을 입증할 수 있는 도구로도 이용될 수 있을 것이다. 그러므로 시뮬레이터의 구성 및 기능도 성능에 비교적 큰 영향을 주는 하부시스템 위주로 모의된다. 시뮬레이터는 일정한 시간 간격마다 discrete한 시각에 동작되고, 하부시스템의 자율적인 동작과 프로그램의 재활용 성을 높이기 위해 객체 지향적 프로그래밍 방법을 사용하였다.

종합시뮬레이터의 프로토타입이 완성되어 한국철도기술연구원에서 현재 개발중인 경량전철시스템 설계의 종합성능 검증에 적용되었다. 본문에서는 전기철도 종합시뮬레이터의 기본 설계개념과 개발된 프로토타입

1. 정회원, 한국철도기술연구원, 책임연구원
2. 정회원, 한국철도기술연구원, 선임연구원
3. 정회원, 한국철도기술연구원, 선임연구원
4. 정회원, 한국철도기술연구원, 책임연구원
5. 비회원, 명지대학교, 교수

을 경량전철시스템 설계검증에 적용한 결과를 간략하게 제시한다. 또한 본문에서 제시되는 시뮬레이터는, 성능 검증이 그 주요한 목적이므로, 시뮬레이터 구성 및 운전방법이 실제시스템과는 성능에 영향이 없는 한 차이가 있을 수 있다는 점을 이해할 필요가 있다.

2. 종합시뮬레이터

종합시뮬레이터에서 각 하부시스템은 각각의 독립된 객체로서 모의된다. 프로토타입 종합시뮬레이터에는 5개의 주요 객체가 있으며 이들은 각각 Operating Control Center(OCC), Automatic Train Protection Equipment(ATP), Station ATO Equipment(Station), Trains(Train), Power DistributionSystem(PDS)를 모의한다. 앞으로는 괄호 속의 약자로 각각의 객체가 지칭된다. 그림1은 시뮬레이터의 구성도를 보여준다. OCC는 주어진 시격에 의해 차량을 배차하며 각 차량의 역 도착시각을 점검한다. ATP는 각 차량의 위치와 속도를 감시하며 각 차량에 제한속도를 부여한다. 열차의 속도는 선로조건, 앞 열차와의 거리를 기준으로 계산된다. Station은 승객의 흐름을 관리하고 열차 Door의 개폐를 명령 감시하며 열차에 출발허가를 부여한다. Train은 ATP로부터 부여된 제한속도 범위내에서 스스로의 열차진행을 조정/수행한다. PDS는 열차로부터 그 위치와 요구전력을 통보 받아 조류해석 및 에너지 분석을 수행한 뒤 각 열차에 전압 정보를 제공한다. 승객은 매 타임 스텝마다 승하차 테이블에 따라 각 역에 나타난다. 각 역에 나타난 승객은 Station 및 Train에 의해 그 흐름이 관리된다. 시뮬레이터 사용자는 수동 모드에서 각 객체의 속성을 바꾸거나 이벤트를 생성시킴으로써 시스템의 운영을 제어 할 수 있다. 그림2는 시뮬레이터의 전체적인 흐름도이며 하부시스템 프로세스들의 동작순서는 데이터의 논리적 순서에 의해 선정되었다.

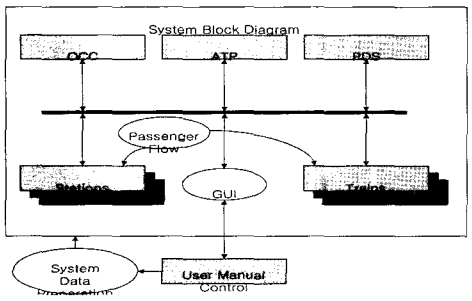

그림 1 시뮬레이터 구성도
Fig. 1 Simulator Basic Structure

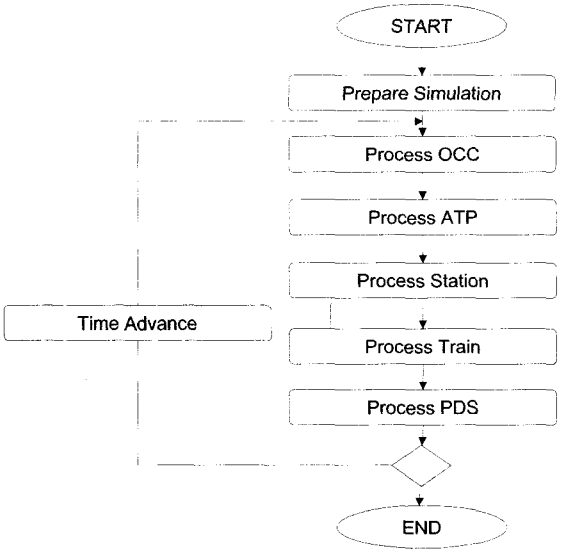

그림 2 시뮬레이션 수행 흐름도
Fig. 2 Simulation Flow Diagram

2.1 OCC

시뮬레이터는 1개의 OCC 객체를 갖는다. OCC는 시격에 따라 열차를 배차한다. 열차 배차 시 OCC는 열차의 bool 속성인 Active를 true로 하며 이것은 그 열차가 운전중임을 의미한다. OCC는 또한 열차 배차 시에 열차의 ID를 일련번호로 부여하고 이 번호를 ATP에 통보함으로써 ATP에게 어느 차량이 운전중임을 알려준다. 열차가 운행을 끝낸 후 회송 위치로 갈 때 열차는 스스로 Active 변수를 false로 만들고 이를 OCC에 통보한다. OCC는 이를 다시 ATP에 통보한다. 열차가 역에 도착하면 Station은 열차의 도착시간을 OCC에 통보하고 OCC는 이를 열차의 계획도착시각과 비교한다. 열차가 계획시각에 비해 허용범위 보다 크게 연착한 경우 OCC는 열차의 최고속도 조정 및 선행열차의 역정차시간 조정을 통해 열차의 지연시간을 줄이기 위한 조치를 취한다[1]. 열차의 최고속도 조정은 ATP에 통보하며 역정차시간 조정은 Station에 통보한다. 표1과 그림3은 OCC 객체와 다른 객체사이의 인터페이스 내용이며 표2는 OCC 클래스의 멤버변수 및 함수이다.

표 1 OCC 인터페이스

인터페이스 객체	방향	인터페이스 내용	내부 기능
Train	To	- 열차배차	- 열차배차 및 통보(ATP) - 열차 운행 스케줄 감시 및 조정
	From	- 열차회송	
Station	To	- 최소 역 정차시간 지정	
	From	- 열차도착시각 통보	
ATP	To	- 운행 중인 열차 ID 통보 - 각 열차의 최고속도지정	

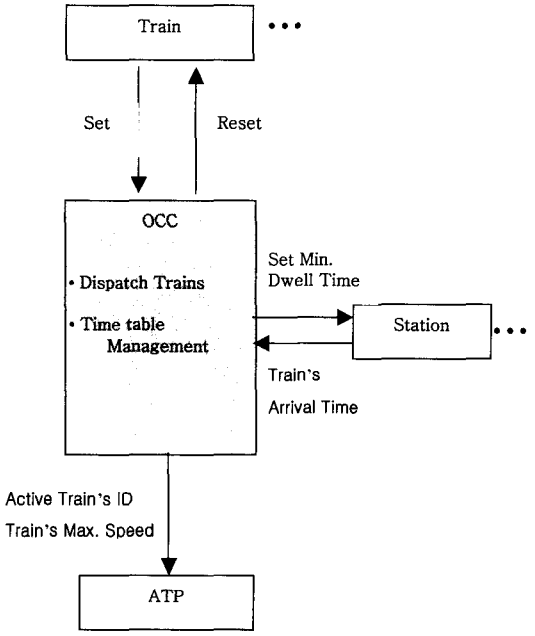

그림 3 OCC와 다른 객체사이의 인터페이스
Fig. 3 OCC's Interface with Other Objects

표 2 OCC 멤버 변수 및 함수

```
class OCCs
{
public:
OCCs();
~OCCs();
void        processOCC();
void        readOCCData();
void        setToReserve(int);
            // 종착역에 도착한 차량-> 기지
int         getUpTrackFirstTrainNo();
           // 상행선 첫 번째 열차번호를 return
int         getUpTrackLastTrainNo();
           // 상행선 마지막 열차번호를 return
int         getDownTrackFirstTrainNo();
           // 하행선 첫 열차번호를 return
int         getDownTrackLastTrainNo();
           // 하행선 마지막 열차번호를 return
private:
int         UpTrackFirstTrainNo,//상행선 첫 번째 열차
            UpTrackLastTrainNo,//상행선 마지막 열차
            DownTrackFirstTrainNo,//하행선 첫 열차
            DownTrackLastTrainNo;//하행선 마지막 열차
float       HeadWay, // 시격
            CrossTime, // 교행시차
            LastUpTrackDispatchTime,
            //상행선 마지막 배차시각
            LastDownTrackDispatchTime;
            //하행선 마지막 배차시각
void        dispatchUpTrackTrain();
            // 상행선 배차
void        dispatchDownTrackTrain();
            // 하행선 배차
void        storeOCCData();
            // OCC 데이터 기록
void        initialize(); // 초기화
};
```

2.2 ATP

시스템은 1개의 ATP를 갖는 것으로 모의된다. OCC는 ATP에 운행중인 차량의 ID를 전달한다. ATP는 ID를 전달받은 차량에 한하여 감시를 수행한다. ATP는 이 ID에 의하여 열차들의 위치와 속도를 감시하며 선행열차의 위치, 선형조건 등을 고려하여 제한속도를 계산하고 이를 다시 각 열차에 통보한다. ATP는 선행열차와의 거리에 의한 최고안전속도, OCC에서 통보된 운행최고속도 및 선형조건에 의한 최고속도의 3 값을

비교하여 가장 작은 값을 열차에 제한속도로 통보한다. ATP는 다음 Time Step에서 열차의 실제운행속도가 부여된 제한속도 보다 클 경우 열차의 bool 변수 Emergency를 true로 만들어 열차를 비상제동 시킨다. 표 3과 그림4는 ATP 객체와 다른 객체와의 인터페이스 내용이며 표4는 ATP 클래스의 멤버변수 및 함수이다.

표 3 ATP 인터페이스

인터페이스 객체	방향	인터페이스 내용	내부 기능
Train	To	- 제한속도 지정 - 비상제동 지시	- 열차 안전최고속도 계산 및 통보 - 열차위치 및 속도 감시 - 비상정지 수행
	From	- 열차속도, 위치, 다음역사 ID 확인	
OCC	From	- 운행중인 열차 ID 통보 - 각 열차의 최고 속도 지정	

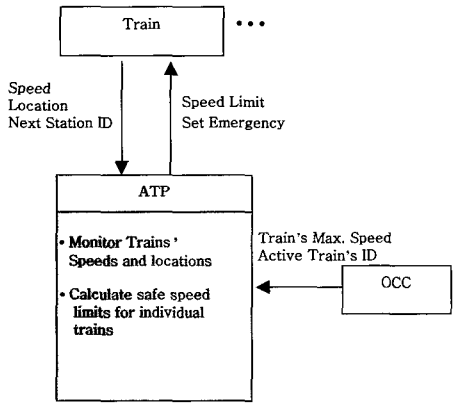

그림 4 ATP와 다른 객체사이의 인터페이스

Fig. 4 ATP's Interface with Other Objects

표 4 ATP Class 멤버 변수 및 함수

```
class ATPs
{
public:
void processATP();
void setUpTrackFirstTrainNo(int );
          //상행선 첫 열차 지정
void setDownTrackLastTrainNo(int );
          //하행선 마지막 열차 지정
void setUpTrackLastTrainNo(int );
          //상행선 마지막 열차 지정
void setDownTrackFirstTrainNo(int );
          //하행선 첫 열차 지정
private:
int UpTrackFirstTrainNo,// 상행선 첫 열차
    UpTrackLastTrainNo,// 상행선 마지막 열차
    DownTrackFirstTrainNo, //하행선 첫 열차
    DownTrackLastTrainNo; //하행선 마지막 열차
    ATPTrains ATPTrain[MAXTRAIN];
          // ATPTrain Array
void initializeATPTable(); // 초기화
void initializeATP(); // 초기화
void getTrainData(); //열차의 현재 위치와 속도
void checkUpTrackTrainData();
          // 제한속도 위반 여부 점검
void checkDownTrackTrainData();
          // 제한속도 위반 여부 점검
void setTrainData();
          // 제한 속도 및 비상정지
void calculateATPSpeed();
          // 제한속도 계산
void storeATPData(); // ATP 정보를 저장
float findTrackSpeedLimit(float);
          // 선로조건에 의한 제한속도
};
```

2.3 Station 객체

시스템은 실제 역사 수와 동일한 수의 Station 객체를 갖는다. 열차가 역에 도착하면 Station의 bool 속성인 "Occupied'를 셋 함으로써 Station에 열차가 도착되었음을 알린다. 이 때 열차의 ID와 승객 수를 함께 통보한다. Station은 이 도착정보를 OCC에 통보하여 OCC가 열차 운행시간 조정/관리를 할 수 있도록 한다. 다음 Time Step에서 Station은 열차 Door Open 지령을 전달한다. 다음 Time Step에서 열차의 Door Open을 확인 후에 승객하차 프로세스를 시작한다. 승

객 흐름의 관리는 Station에서 수행된다. 다음 Time Step에서 승객하차 프로세스가 완료되었으면 승객의 승차 프로세스를 시작한다. 다음 Time Step에서 승객 승차 프로세스가 완료되고 열차의 역 정차시간이 최소 역 정차시간보다 크면 Station은 열차에 출발허가를 한다. 열차는 출발허가를 받고 ATP로부터의 제한속도가 0보다 크면 출발을 한다. 이때 열차는 Station의 'Occupied' 변수를 리셋 함으로써 Station에 열차가 출발되었음을 알린다. 열차가 역에 도착 시에 열차는 열차내의 총 승객 수를 Station에 통보한다. Station은 총 승객수로부터 다음의 방법으로 당 역 도착 승객 수를 계산한다. Station은 입력된 승하차인원 자료로부터 식(1)에 의해 Unload Factor를 계산하여 갖고 있으며 식(2)에 의해 당 역 도착 승객 수를 계산한다. 표 7은 승하차 테이블의 예로서 계산된 Unload Factor를 함께 보여준다. 4번역의 Unload Factor는 13.06%이며 이것은 4번역에 도착한 열차 총 승객수의 13.06%는 4번역에서 하차함을 의미한다. 승객은 매 Time Step 역에 출현하므로 역대기승객 수는 매 Time Step 증가한다. 매 Time Step 나타나는 승객의 수는 승하차 테이블에서 구해진다. 승객이 열차에서 하차하는 속도와 승차하는 속도는 열차 Door의 크기와 수에 대체적으로 비례하는 것으로 보고되고 있으며[2] 입력 데이타로 주어진다. 열차는 항상 승객의 수를 점검하고 있으며 승객의 수가 열차의 최대탑승인원 수와 같아지면 열차는 만차가 된다. 열차가 만차가 되거나 대기승객의 수가 0이 되고 열차의 당 역 정차시간이 최소정차시간보다 크면 열차의 승차프로세스는 완료된 것으로 간주되고 Station은 다음 Time Step에서 출발허가를 한다. 이때 다음 역의 ID를 함께 열차에 통보한다. 표 5와 그림5는 Station 객체와 다른 객체와의 인터페이스 내용이며 표6은 Station 클래스의 멤버변수 및 함수이다..

$$UF_i = DN_i \times 100 \div \left(\sum_{k=0}^{i} ON_k - \sum_{k=1}^{i} DN_k \right) \quad \text{---------- (1)}$$

UF_i : Station i의 Unload Factor

ON_k : 승하차 테이블에서 Station k에서의 승차 승객수

DN_k : 승하차 테이블에서 Station k에서의 하차 승객수

$$AP_i = UF_i \times TP_i \div 100 \quad \text{---------- (2)}$$

AP_i : Station i에서 하차하는 승객 수

TP_i : Station i에 도착한 열차의 총 승객 수

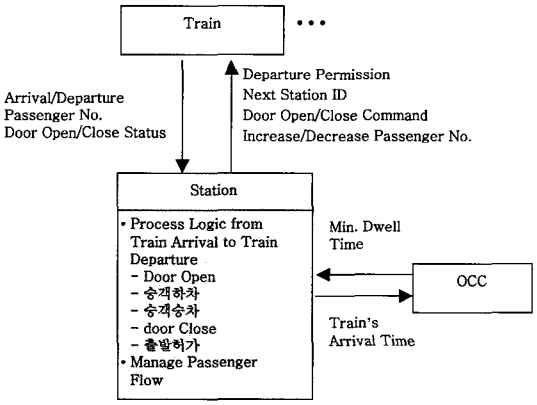

그림 5 Station과 다른 객체사이의 인터페이스

Fig. 5 Stations', Interface with Other Objects

표 5 Station 인터페이스

인터페이스 객체	방향	인터페이스 내용	내부 기능
Train	To	- Door Open/Close 지시 - Door Open/Close 확인 - 승객 승차 및 하차 - 다음 목적지 통보 및 출발허가	- 열차 도착 승객수 계산 - 승객 하차 - 승객 승차 - 열차출발허가 - 대기승객 증가 - 열차 도착시각 통보(OCC) - 역 최소정차시간 관리
	From	- 열차도착통보 - 열차 ID 및 총 승객 수 통보	
OCC	To	- 열차 도착시각 통보	
	From	- 최소 역 정차시간 지정	

표 6 Station Class의 멤버 변수 및 함수

```
class Stations {
public:
Stations();
~Stations();
float    getLocation(); // 역사 위치
void     setLocation(float); // 역사위치
void     setDwell(float); // 최소 역정차시간
void     setWPassengerIncreaseRate(float);대기승객
void     setUnloadFactor(float ); // 하차 율
void     setId(int); // 역사 ID 지정
void     setArrivingPassengerNo(int );
void     setDoorOpen();
void     setDoorClose();
void     processStation();
int      getWaitingPassengerNo();//역 대기승객수
int      getWPassengerIncreaseRate();//역대기승객증가율
bool     isTrainInStation();
void     setTrainInStation(int); // 열차 역에 정차
void     resetTrainInStation(); // 열차 역에서 출발
void     initializeAgain(); 초기화
void     setLoadingRate(); // 승객의 차량 승차속도
void     setUnloadingRate(); // 차량으로부터 하차 속도
void     printStationPassenger(); //
private:
int      Id,
         TrainId, //도착하고 있는 열차 번호
         ArrivingPassengerNo, // 도착 차량이 assign
         MaxWaitingPassengerNo, //최대 역 대기승객
         UnloadingRate, // 초당 차량에서 하차하는 승객수
         LoadingRate, // 초당 차량에 탐승하는 승객수
         UnloadFactor;
float
         ArrivalTime, // 열차의 역 도착시각
         DepartureTime, //열차의 역 출발시각
         MinDwell, // 최소 역 정차시간
         DwellTime, // 역 정차시간
         MaxTime, //
         WaitingPassengerNo,
         WPassengerIncreaseRate, //역 대기승객 증가율
         Loc; 역 위치
bool     TrainInStation, // one train stays in the station
         DoorOpen, // 차량 Door Open 확인
         LoadingEnd, // 승차완료
         UnloadingEnd; // 하차 완료
void     createPassenger(); 역 대기승객의 증가
void     givePermissionToLeave(); // 열차 출발허가
void     unloadPassenger(); 승객 하차시킴
void     loadPassenger(); 승객 승차시킴
};
```

표 7 승하차 테이블 예와 Unload Factor

역사번호	Origin No./hr	Destination No./hr	Unload Factor [%]
0	522	0	0
1	342	14	2.68
2	456	34	4.0
3	612	184	14.47
4	245	222	13.06
5	154	658	38.19
6	122	532	43.64
7	58	448	55.38
8	66	362	86.4
9	7	88	71.54
10	0	42	100.
계	2,584	2,584	

2.4 Train

OCC는 시격에 의해 열차를 배차할 때 열차의 bool 속성 'Active'를 셋 한다. 또한 일련번호로 ID를 부여한다. Active 변수가 셋되어 있는 열차만 매 Time Step 운전이 모의된다. 열차가 종착역에 도착하여 회송기지로 향해 출발 할 때 열차는 스스로 Active 변수를 리셋한다. 또한 이를 OCC에 통보한다. 역과 역 사이를 진행할 때 열차는 3 가지 타입의 구간을 통과한다. 첫째는 역사 내 구간이다. 여기서는 Station으로부터 출발허가를 기다리고 있는 상태다. Station으로부터 출발허가를 받고 ATP로부터의 제한속도가 0보다 크면 열차는 출발하고 그 다음 구간인 역 사이 구간으로 들어간다. 역 사이 구간에서 열차는 ATP의 제한속도와 열차의 동력한계 범위 내에서 스스로의 가속도를 결정한다[3]. 먼저 열차는 ATP 제한속도 보다 3km/h 작게 목표속도를 설정한다. 그리고 현재속도가 목표속도보다 2km/h 이상 작으면 가속한다. 만약 현재속도가 목표속도의 ±2km/h 범위 내에 있으면 코스트 운전을 수행하고 목표속도보다 2km/h 이상 크면 감속한다. 열차가 트랜스폰다를 지나면 역 진입 구간으로 들어가며 이 구간에서는 열차의 감속도가 -2.5km/h 부근으로 일정하게 역 도착 시까지 운행된다. 열차 Emergency 변수가 셋 되어 있으면 운행되고 있는 구간에 관계없이 열차는 비상제동 감속도로 운전된다. 열차의 가속도나 감속도가 결정되면 열차는 요구전력을 계산하고 그 값을 열차위치와 함께 PDS에 통보하고 PDS는 열차의

가선전압을 열차에 다시 통보해 준다. 열차의 가선전압이 허용범위 이하로 내려가면 열차는 얼람을 발생시킨다. 표 8과 그림6은 Train 객체의 다른 객체와의 인터페이스 내용이며 표 9, 10은 Train 클래스의 멤버변수 및 함수이다..

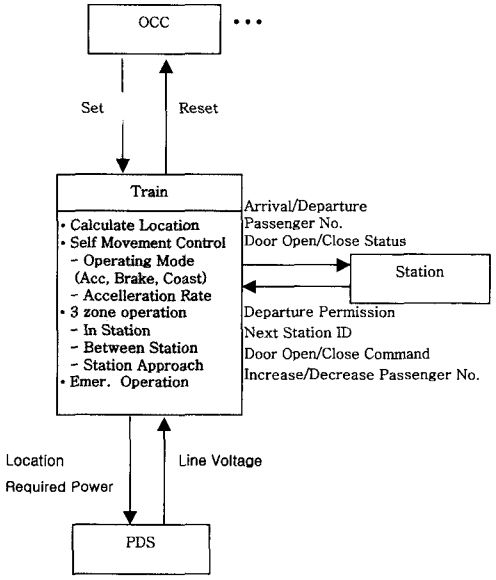

그림 6 Train과 다른 객체사이의 인터페이스

Fig. 6 Trains' Interface with Other Objects

표 8 Train 인터페이스

인터페이스 객체	방향	인터페이스 내용	내부 기능
Station	To	- 열차도착통보 - 열차 ID 및 총 승객 수 통보	- 열차 운행 운행구간 확인 운전모드 선정 운전속도 계산 - 역 정차 및 출발 열차 DoorOpen/Close - 열차 승객 수 관리
	From	- Door Open/Close 지시 - Door Open/Close 확인 - 승객 승차 및 하차 - 다음 목적지 통보 및 출발허가	
ATP	To	- 열차속도, 위치, 다음 정차 역 ID 확인	
	From	- 제한속도 지정 - 비상제동 지시	
OCC	To	- 열차 회송 통보	
	From	- 열차 배차	
PDS	To	- 위치 및 요구전력 통보	
	From	- 열차 전압 통보	

표 9 Train Class 멤버변수 및 함수(1)

```
class Trains
{
public:
Trains();
~Trains();
void processTrain();
void setEmergency();
void setPermitToLeave(int); // 출발허가
void setInStationRegion(); // 역사 정차 중
void setATPSpeed(float); // 제한속도 지정
void setDoorOpen(int); // 차량 Door Open
void setDoorClose(int); // 차량 Door Close
void setArrivalTime(float); //종착역 도착시각
void storeTrainData();
void decreasePassengerNo(int); //
void increasePassengerNo(int); //
void setId(int);
void initializeUpTrack(int); //
void initializeDownTrack(int); //
int   getPassengerNo(){return PassengerNo;}
int   getMaxPassengerNo(){return MaxPassengerNo;}
int   getId(){return Id;}
float getDepartureTime(){return DepartureTime;}
float getArrivalTime(){return ArrivalTime;}
float getTravelTime(){return TravelTime;}
float getLocation(){return Loc;}
float getSpeed(){return Speed;}
float getPower(){return Power;}
private:
int         Id,
            DStaId, //다음 도착 역의 ID
            PassengerNo, // 열차 내 승객수
            MaxPassengerNo; // 열차 내 최대 승객 수
float       Speed, // 열차 속도
            AverageSpeed, // 열차 평균속도
            Loc, // 열차 위치
            DLoc, // 다음 도착 역사의 위치
            TLoc, // 트랜스폰다 위치
            Acc, // 열차 가속도
            TMass, // 열차 질량
            DyMass, // 열차 관성질량
            Power, // 열차 요구전력
```

표 10 Train Class 멤버변수 및 함수(2)

```
계속
private:
float       OldAcc, // 전 스텝에서 열차 가속도
            ATPSpeed, // ATP로부터 제한속도
            TSpeed, // 열차 목표속도
            AForce, // 열차 출력 가능한 최대 힘
            TResist, // 열차 Total 저항
            DepartureTime, // 열차 시발역 출발시각
            ArrivalTime, // 열차 종착역 도착시각
            TravelTime; // 열차 운행시간
char Mode; // 열차 운전 모드( 가속, 타행, 감속, 정차)
bool
            DStop, // 열차 Dead Wall Stop
            PermitToLeave, // 열차출발허가
            Emergency, // 비상정지
            InStation, // 열차 역 정차
            InStationRegion, // 역 접근 구간
            Active, // 열차 운행 중
            DoorOpen; // 열차 문 열림
void processArrival(); // 열차가 역 도착 시  로직
void processBetweenStations();// 역 사이 구간
void processEmergency();// 비상정지 수행
void processDeparture();// 열차 역 출발
void processOthers();
void selectMode(); 열차 운전 모드 선택
void notchUp(); 가속
void coastOperation(); 타행운전
void brakeOperation(); 감속운전
void moveTrain(); 열차 이동
void findPower(); 열차 요구전력 계산
void findTotalResist(); Total 저항 계산
void findAvailForce(); 열차 가능한 최대추진 힘 계산
void findTLocation(); // 트랜스폰다 위치 얻음
void checkTLocation(); // '트랜스폰다 위치 지남?'
int findCurveData(); // 현 위치의  curve data 구함
int findGradData(); // 현 위치의  구배 data 구함
float findCurveResist(); // 현 위치에서의  curve 저항
float findGradResist(); //  현 위치의  구배 저항 구함
float findRunningResist(); 주행저항 구함
};
```

2.5 PDS

PDS 1개의 객체로 모의된다. PDS는 열차로부터 위치와 요구전력을 통보 받아 전력조류해석을 수행하고 전차선 전압을 각 차량에 통보한다. 전차선의 전압이 회생전력에 의해서 허용전압 이상으로 상승할 경우 차량은 회생제동을 포기하고 기계식 제동장치에 의해서 제동되는 것으로 모의되었다. 개발되는 경량전철시스템에서 차량 최대허용전압은 정격전압의 120%로 모의되었다. 그림 7은 PDS 객체의 다른 객체와의 인터페이스 내용이며 표 11, 12는 PDS 클래스의 멤버변수 및 함수이다.

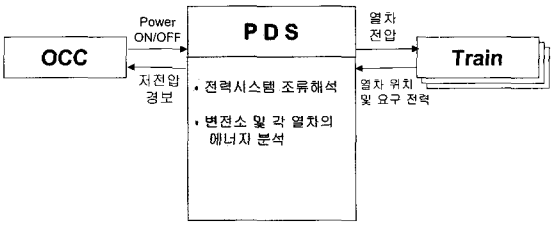

그림 7 PDS와 다른 객체사이의 인터페이스
Fig. 7 PDS's Interface with Other Objects

표 11 PDS Class 멤버변수 및 함수(1)

```
class PDSs{
public:
PDSs();
~PDSs();
void setUpPDS();
void readSubData();
void processPDS();
private:
int         NodeNo, // 노드 수
            StaNo, // 역사 수
            LineNo, // 라이 수
            Iter, // 반복회수
            From[MAXLINE], 라인 시작 노드번호
            To[MAXLINE]; 라인 끝 노드 번호
float       Location[MAXSTEP], 노드 위치
            G[MAXNODE][MAXNODE],
            I[MAXNODE],
            V[MAXNODE],
            OldV[MAXNODE],
            Cond[MAXLINE], // 라인 컨덕턴스
            Curr[MAXLINE], // 라인 전류
            VNoload, //무부하 전압
            VBase,
            RBase,
            IBase,
            ResPerKm, // 전차선저항/km
            Tolerance, //에러허용값
            AuxPwr, // 보조전원 요구전력
            MaxTrainVolt, // 차량최고전압
            TrainRes; //열차 내부저항
bool        StatusChange; // 전원 혹은 차량 모의변경
```

표 12 PDS Class 멤버 변수 및 함수(2)

```
계속
void initializeNodes();
void calculateBase();
void buildNode(); // 노드 선정
void buildSubNode(); // 변전소 노드 선정
void buildStationNode(); // 스테이션 노드 선정
void insertNode0(int ,int, char );
void insertNode1(int ,int, char );
void buildTrainNode(); // 차량 노드 선정
void buildConnectionData(); // 노드 사이 연결
void buildG(); // [G] 매트릭스 형성
void buildG1(); // [G] 매트릭스 재형성
void buildI(); // [I] 벡터 형성
void decompLL(); // [L][L']로 Decompose
void modifyI(); //[I] 벡터 수정
void solveLL(); //
int withinTolerance(); // '에러가 허용범위 내?' 검토
void initializeV();
void convertAll(); // 회생차량부하 모두 전압원
void checkSub(); // 변전소 전압과 시스템 전압 비교
void checkTrain1(); // 차량전압과 차량최고전압 비교
void checkTrain(); // 차량전압과 차량최고전압 비교
void loadflow(); // 조류해석
bool onlyVSource(int); // '유일한 전압원?' 검토
void initializeSubStatus();
void initializeSub1();
void updateGTrainGStationData(); //데이타 기록
void substationRecord();//데이타 기록
void calculateLineCurrent(); // 라인 전류 계산
struct subs sub[MAXSUB];
struct nodes node[MAXNODE];
};
```

3. 사용자 인터페이스

그림8은 메인 화면을 보여준다. 'Start' 버튼을 누르면 자동운전모드로 시뮬레이션이 시작된다. 옆의 'Manual'버튼은 수동모드의 시뮬레이션을 위한 것이다. 'Start'버튼 왼쪽에 4개의 버튼이 있으며 각각 'OCC', 'ATP', 'Station', 'Train'의 4 타입 객체의 상세 운전상태를 감시하기 위함이다(그림 9). 'Manual' 버튼을 누르면 사용자가 각 객체의 속성을 바꾸어 가면서 수동운전을 할 수 있다(그림10, 11). 사용자는 또한 타임 다이아그램(그림12)을 통하여 각 열차의 스케듈 대비 운행 현황을 감시할 수 있다. 메인화면 상단 우측의 'Graph'화면에서는 관심있는 차량을 선택해서 속도-거리 특성을 감시할 수 있다.(그림13) 시뮬레이션이 진행되는 동안 각 열차의 진행과 전차선 전압의 변동이 메인 화면에 보여진다(그림8). 열차의 색깔은 열차의 운전모드에 따라 달리 표시된다. 빨간색은 브레이킹 모드, 가속의 경우 파란색, 코스팅 모드는 초록색, 역 정차 시에는 노란색으로 각각 표시된다.

4. 시험적용

개발된 시뮬레이터를 철도기술연구원에서 개발중인 경량전철시스템의 성능에 시험 적용하였다. 경량전철시스템은 목표노선을 상정하여 개발되고 있다. 목표노선의 승하차 테이블 중 최악조건 시간대의 승하차 테이블은 표 13과 같다. 시뮬레이션은 각각 시격 120초와 90초를 적용하고 현재 승하차 인원수요와 향후 수송수요의 증가에 대한 오차를 고려한 120% 승하차 인원수요에서 수행되었다. 표14는 시뮬레이션 결과의 요약을 보여 준다. 시뮬레이션의 목적은 편성 당 적정 차량 수를 결정하기 위함이다. 본원에서 개발되고 있는 경량전철은 2량 1편성이 기본단위로 증설 편성 시에는 2량씩 증편할 수 있다. 편성 당 차량 수는 가급적 작을수록 플래트폼의 길이를 줄일 수 있어 역사 규모의 축소, 도입차량 수의 감소 등 초기투자비를 줄일 수 있는 이점이 있다. 표14에서 열차최대탑승인원은 시뮬레이션 기간 중 가장 승객이 많은 편성의 승객 수이다. 열차는 정원의 200%까지만 탑승할 수 있도록 모의되기 때문에 열차 수송 능력이 수요를 능가하지 못하면 역 대기승객은 점점 증가한다. 표 14에서 보면 100% 승하차 인원수요를 적용하였을 경우, 혼잡도 허용 값을 200% 기준으로 판단하면 120초 시격에서 4량 1편성은 큰 무리 없이 수송수요를 만족하고 있다. 120% 승하차 인원수요를 적용하였을 경우 4량 1편성은 120초 시격에서 허용 값을 200%를 상회하고 역사 4의 대기승객이 지속적으로 증가하여 1600초 시뮬레이션기간 중 역사 4의 대기승객이 초기 값 50명에서 304명으로 증가하였고 계속 증가할 것이다. 6량 1편성의 경우 120% 수송 수요에서도 쾌적한 승차환경이 제공된다. 원래 시스템 요구조건의 승하차 인원수요는 4량 1편성으로 만족한다. 120% 승하차 인원수요에서의 검토는 시스템의 수송 여유 분의 확보를 위함이다. 또한 시격을 90초로 단축시켜 운행한다면 120% 승하차 인원수요도 충족시킬 수 있다. 엔지니어링 측면에서는 4

량 1편성, 6량 1편성 모두 타당성이 있다. 승객에 보다 쾌적한 승차환경을 제공하고 미래의 불확실성에 대한 대비를 위해서는 추가 가격을 지불해서 6량 1편성으로 갈 수도 있다. 정책적 판단이 될 것이다.

표 13 목표노선의 승하차인원 테이블

역사번호	승차인원/hr	하차인원/hr
0	3600	0
1	2400	100
2	3000	200
3	3000	900
4	1800	600
5	1200	3600
6	400	3600
7	400	3600
8	400	2400
9	400	1200
10	0	600

표 14 목표노선의 운행시뮬레이션 결과, 1600초 시뮬레이션

수송수요	편성당 차량수	120초 시격			90초 시격		
		열차최대탑승인원	혼잡도 %	최대역사대기승객	열차최대탑승인원	혼잡도 %	최대역사대기승객
100% 승하차 인원수요	4	416	182.5	102	303	132.9	72
	6	406	118.7	102	303	88.6	72
120% 승하차 인원수요	4	456	200.0	304	367	161.0	84
	6	493	144.2	120	363	106.1	84

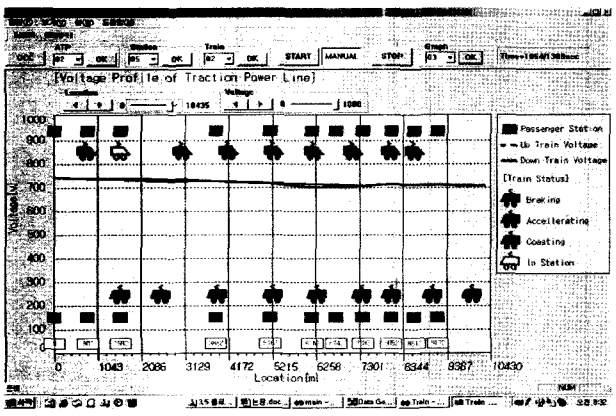

그림 8 시뮬레이터 메인 화면

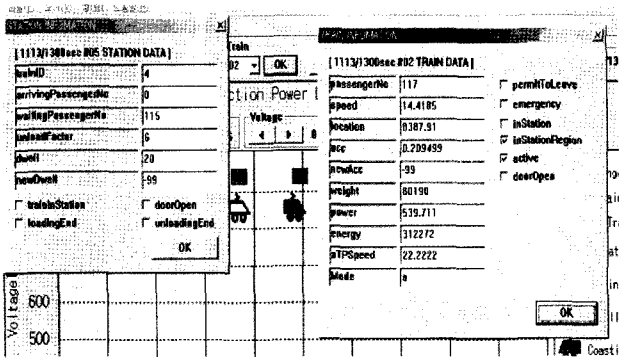

그림 9 객체속성 상세 표시

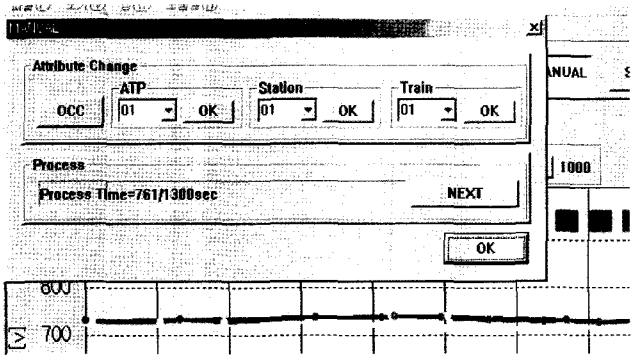

그림 10 속성 변경을 위한 객체선택

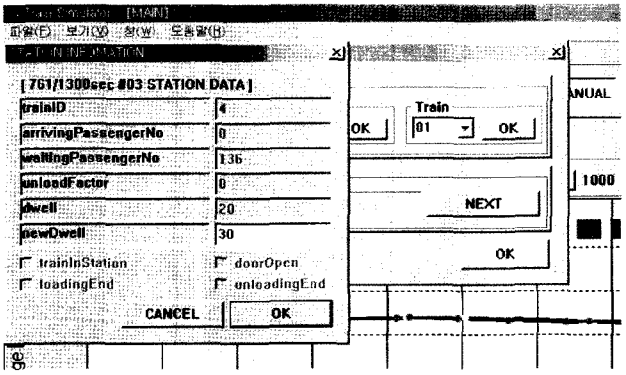

그림 11 객체 속성 변경

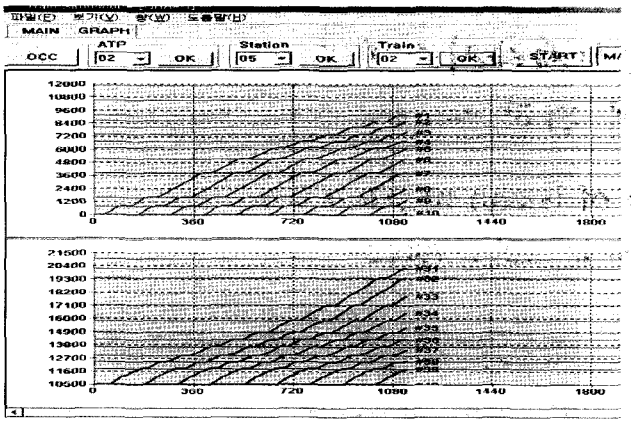

그림 12 타임 다이아그램

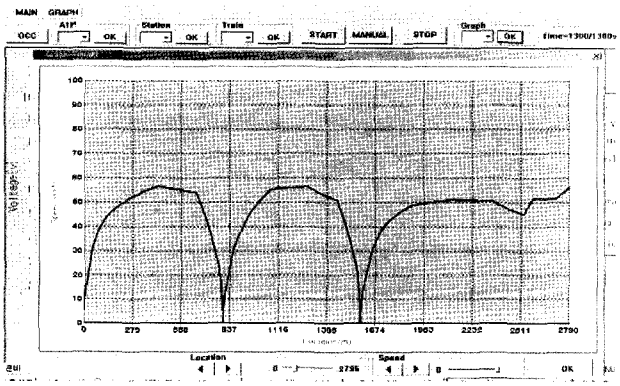

그림 13 차량 객체의 속도-거리 특성

6. 결 론

종합시뮬레이터의 특징은 여러 하부시스템이 동시에 모의되는데 있다. 시험적용 예와 같이 승객의 흐름에서도 모든 다른 하부시스템이 영향을 줄 수 있다. OCC는 시격을 조정함으로써, ATP는 제한속도를 조정함으로써, 역사는 역 정차시간을 조정함으로써, 또한 역사 대기 승객의 수를 조정함으로써, 차량은 역간 운행시간을 조정함으로써 승객의 흐름은 변한다. 즉 도시철도 하부시스템은 각각 서로 밀접한 영향을 주면서 동작한다. 이러한 하부시스템을 동시에 모의함으로써 보다 현실적인 모의가 가능하다. 단일분야의 시뮬레이터 경우는 다른 분야의 상태는 대개 최악조건의 상태로 가정된다. 시스템 설계에 있어 이 방법은 안전한 결과를 줄 수 있지만 최적의 값을 주지는 못한다. 종합시뮬레이터는 또한 여러 분야가 포함된 문제를 파악하고 개선 방안을 줄 수 있으므로 시스템 연구개발에도 유용한 도구로 사용될 수 있을 것이다. 개발된 시뮬레이터는 주로 한국철도기술연구원에서 개발중인 경량전철시스템의 특성이 모의되었다. 또한 객체지향 프로그래밍을 이용하여 다른 시스템을 모의 할 때도 기존 프로그래밍의 재활용이 용이하다. 하부시스템의 특성이 다를 경우 그 하부시스템의 특징만 다시 모의되면 되기 때문이다. 향 후 더 진전된 시뮬레이터로서는 모든 하부시스템이 하드웨어상의 부속품처럼 콤포넌트화 되어 서로 다른 콤포넌트를 붙였다 띠었다 해 가면서 모의할 수 있는 시뮬레이터가 가능할 것이다.

1) M.Kanayama, M.Miyoshi, Y.Seki Y.Fujiwara, K.Sobu, 1998, "Development of Train Control Simulator", International Conference on Developments in Mass Transit Systems, 20 23 April 1998

2) 한국철도기술연구원,"인천국제공항 IAT시스템 구축사업 제안요청서", vol.2 STS160, 2003. 2.

3) 한국철도기술연구원,"경량전철기술개발사업 3차년도 연구결과보고서(종합시스템엔지니어링분야)", 페이지 213, 2001. 12