Object oriented simulation of electric network for urban transport

F. Gagliardi, L. Ippolito & A. Piccolo

Abstract

Recently, some authors have proposed the use of Object Oriented Programming (OOP) procedure for the simulation of the railway signalling and train movement.

The application of above mentioned procedure to the railway electric network seems very easy and at the same time very effective. In fact the basic idea of the OOP is an object which is thought as an abstraction with boundary and meaning linked to the specific problem in study, the traction system is naturally subdivided in objects having network structures such as railway network topology, the OOP permits to model in a simple and suitable way the interconnections among subsystems, the OOP permits to easily operate a large amount of data.

However particular attention has to be paid to the modelling of the objects, especially the train and its drives. Modelling inaccuracies of these components are not evidenced by the OOP procedure and they have a relevant weight on the results obtained by the simulation.

In the paper, at first a brief review of OOP concepts is recalled, paying attention to ones relevant for the electric supply of urban and suburban railways.

Successively, the identification and the steady state modelling of the significant components of fixed plants are studied.

The adopted architecture is fitted with the OOP procedure and the chosen language is the Visual C++.

Particular attention is paid to the specific hypothesis, because they strongly constrain the results accuracy and their reliability.
1 Introduction

The software represents today a useful tool in order to describe the behavior of a more or less complex real system. A software code can be seen like an operational model of the whole or part of the physical system in study. When the software design is so thought, the OO design results to be the most natural approach: the physical system is made of objects, and so the creation of the models based on a software representation of such objects appears logical. In the analysis of the electric network for urban transport, physical characteristics of the traction system satisfy naturally to such a principle, justifying an Object Oriented (OO) approach, in which the objects is seen like an abstraction with boundaries and meanings typical of the system in study (Siu & Goodman [1], Galaverna & Sciutto [2], Pozzobon & Sciutto [3], Okumura & Ishida [4], Battistelli, Gagliardi, Ippolito & Piccolo [5]). In the paper the base rules that allow to build an OO software simulator - written in Visual C++ - for the train movement and railway network, are at first described. Particular attention is paid both to the design software aspects and to the models physical aspects. Only the knowledge of detailed and precise analytical models of the traction system different objects makes possible to proceed to the implementation of a simulator able to furnish reliable results and to overcome the phase of validation on the field, whose importance appears today more and more remarkable. For these reasons in the paper the models chosen for the motors and the electric network are examined in detail.

2 General rules for classes development

It is well known, that in the design of a software simulator of electric network for urban transport it is necessary at first to find the subsystems objects, in which the traction system can be divided. Once it is settled on the kinds of objects, these have to be described. The OO languages satisfy naturally this requirement by class idea, which allows to describe data structures characterized by common property (Mayer [6]).

On the other hand, in the present case the class development have to guarantee:

- the respect of the contractive efforts and the complete exploitation of the OOP facilities;
- an exhaustive description of the evolution of the quantities investigated;
- an efficient exchange of information among the various classes (messages);
- a contained computational effort;
- the portability of a wide part of code.
In base to the preceding considerations, it is possible to reach the individuation both of some basic classes, to which entrust the definition of a set of messages able to guarantee the solution of the problem in the most general terms, and of some derived classes, to which assign the detailed description of the solution techniques. In such a way it is possible to obtain the definition of a classes library, more extensible and portable as possible.

To allow a better exploitation of the Windows environment, selected as the simulator interface, a mechanism of the compound derivation is used to reach classes that inherit at the same time the code of each of the classes of the library and of the CObject, a Windows predefined class. This mechanism allows to gain some important features:

1. the serializability and the persistence (ability to reply to Windows messages of file saving at any moment);
2. the availability of class run-time informations (about class name, object size, class hierarchical position, etc);
3. the diagnostic and the dump (possibility to test the integrity of an object, to verify the consistency of his inner state, to display these informations during the debugging).

Anyway, during the creation of the classes library, particular attention must be paid to those classes describing the trains and the electric network, as the modeling of these objects is more difficult.

3 Train Modeling

To keep the possibility of future changes in terms both of train descriptive profile or of the motor solution techniques, it is better at first to identify the changing parts of the train, and to move the hierarchy design from this decomposition. Therefore, the train system is divided in three modules (vehicle, motor and control logic), each corresponding to a particular activity and interacting in continuous mode with the operator.

From an electrical point of view, the vehicle module allows to represent a train like an electrical dipole that moves. It exchanges with the external modules only three electric quantities: the catenary voltage, the fed current and the phase angle between voltage and current. To these electrical quantities, it is necessary to add the train location (to complete the network solution), the instantaneous speed (to solve the motor equations), and the commands (to manage the network traffic and the signalling). In definitive, the vehicle module is treated like a system that in input accepts a voltage $V_c$ and a commands state, and that in output produces a current $I$, a phase angle $\delta$ between voltage and current and all the informations about its instantaneous position and speed, evaluated in base to the informations contained in its inner state.
The motor module allows to evaluate the current $I$ and phase angle $\delta$ beginning by the knowledge of $V_c$ and commands state. Obviously, such an evaluation requires the solution of the motor equations. This last is surely one of the entities more subjected to changes, in consequence of different situations or of solution different algorithms, therefore it has been chosen to represent it like a logical module separated from the train one. This choice permits to achieve some important advantages:

- to exploit the mechanisms of the inheritance between the classes in order to build representations gradually more complex, recovering code from preexistent classes, without redesign the module that represents the vehicle;
- to use always the same module for the vehicle changing only the motor module.

The motor module consents, instant for instant, to fix a relation between the main voltage $V_m$, the frequency $f_m$ and the instantaneous speed, in input, and the fed current $I_m$, the phase angle $\delta$ between voltage and current and the tractive effort $F_m$, in output. Nevertheless, even if theoretically the considered quantities are sufficient to obtain a solution, really the presence on the motors of starting resistors complicates considerably the equations solution, producing an increase of the required computational time. It is then preferred to add in input another quantity, that represents the required tractive effort $F_r$, furnished from the vehicle module; in such a way the starting resistors are included inside the motor module, obtaining a lower computational time, a more flexible interface and a better simplicity of the total structure.

The logic control module represents the interface between the vehicle module and the motor one, it allows to realize a suitable transformation of the quantities in input. At first, it receives in input some quantities, like catenary voltage, speed, effort, position, etc., and it makes them compatible with the motor module; afterward it receives from the motor module the quantities calculated and it converts these last, according to the set of the motor present on board, giving back the results to the vehicle module.

The until here presented whole information interchanges can be summarized in the following way:

![Fig. 1 - Information interchange system](image-url)
The effected subdivision in modules of the train system allows to create the hierarchy of classes shown in fig. 2.

![Train class hierarchy](image)

**Fig. 2 - Train class hierarchy**

The **ZVehicle** class is a basic class in which the main characteristics of a generic train are contained. The member functions of this class, more than those for the creation and the destruction, will be:

- **Edit**: it edits the derived classes;
- **Reset**: it resets the inner state of the object (stop state);
- **Update_State**: it manages the module train during the simulation;
- **Control_State**: it tests the state consistency;
- **Get_State**: it returns a structure with all the quantities of interest;
- **Set_Command**: it gets from the outside a ZCommand structure type in which the state of the commands is reported. It will be invoked from the network manager, if the train is running in automatic mode, or from a dedicate operator interface function, if the train is controlled in manual mode;
- **Set_Voltage**: it furnishes the value of the network voltage for each train. It is invoked to the end of the electrical network solution to communicate to the trains the calculated voltage at the pantograph.

The functions and the data members of the remaining classes can be written similarly to the ZVehicle class and be summarized in the following manner.

The **ZMotor** class foresees the description of all the messages with which a motor object communicates with the other logical entities, and so it characterizes an interface of general type for the motors. The other classes which describe in particular way the characteristics of each motor and the algorithms of calculus, are derived from the ZMotor class. In particular, the **ZMotor_DC** class, that represents a series excited d.c. motor without starting resistor, the **ZMotor_DC_IND** class, that represents a series excited d.c. motor with management of the starting resistor and of the degrees of field-weakening, the **ZMotor_DC_IND_CP** class, that represents a series...
excited d.c. motor with management of the starting resistor and of the degrees of field-weakening at constant power, are derived. Naturally, they inherit all the members, data and functions, of the ZMotor; nevertheless details in relation to the specific case must be furnished.

The ZLogic class describes the logic control module that has the task to adapt the electrical quantities extracted from the vehicle module in those necessary to the motor, and, in general, to check the quantities of the train relative to the actual running mode (motoring, braking, coasting or stopping mode). The classes, which have in charge the control of motor different kinds (in general it is possible to foresee also motors for not electrified traction) are derived from the ZLogic class. The ZLogic_DC_IND class is an example of a class derived from ZLogic, it permits the solution of the specific case of d.c. series motor drives, which are directly connected to the electric network. The solution foresees the adoption of a precise sequence of operation modes, in each of which the connections among the motors and the field-weakening factors change. The informations have been represented by vectors, readable by means of an index that represents the actual mode of operation.

Since it is necessary to represent many characteristics of the real objects by means of relations not always expressible in closed form, at least two special data structures are required: ZFPointArray and ZFloatArray. All the above mentioned classes have been written under the respect of the standards ANSI C++ and they result perfectly portable toward any hardware platform on which a C++ compiler is installed.

4 Electric and Fixed Plant Modeling

In order to describe the network, fixed plant and feeding catenary, a node-link type approach, appropriately modified according to the OOP methodology, can be still used. For the implementation it is useful to distinguish four different classes, according to the hierarchy shown in fig. 3.

![Network class hierarchy](image)

Fig. 3 - Network class hierarchy

An object belonging to the ZNode class represents an electrical connection between two or more network links. If one or more links present a zero resistance then the correspondent nodes will be shorted. Therefore
the number of nodes, for which the voltage measure occurs during the simulation, can be less or equal to the number of ZNode objects. It is interesting to observe that the objects of ZNode characterize only the connections relative to the fixed topology of the network, so they are completely distinct from the moving nodes of the network, representing the trains connections. Relatively to data members it has been foregone:

- the values of voltage and of phase, in order to characterize electrically the node;
- the table index, to note the connection to the node of an eventual substation;

The **ZSubstation** class consents to describe an electrical substation by means of some electrical quantities (voltage, current, resistance, reactance). Additionally, the presence of an index m_Node allows to locate, on the nodes table, the network node fed by the substation.

The **ZLink** class allows to create link objects, each characterized from an origin node and a destination one, allocated on the nodes table. On the other hand, each network link is described by informations about length, section and resistivity, essential to evaluate the conductances seen from the running vehicles. A corrective factor has been foregone to take into account the case in which the conductors length does not coincide with the horizontal distance between two nodes. The link, which describes a track, is schematized like an ordered sequence of smaller segments, represented by the objects of the ZSegment class. This last contains all the informations relative to the train resistances calculus and to the signalling.

Like already pointed to in precedence, the objects of the **ZSegment** class furnish a complete description of the links. It is, then, possible the representation of segments with the more disparate characteristics as gradients, curves and their radii, stopping points, speed limits, turnout, level crossing, gallery, ecc. In such a way a generic train will be able to use the informations, sent by the network manager, relative to the track that it is crossing. Consequently the running mode can be changed in function of the maximum running speed fixed for that specific track or of other parameters that characterized the segment.

The **ZTab_Res_Moto** class allows to create an object containing the train resistance table relative to each train present. Like already observed, such tables allow to determine the train total resistance for each train that moves on the network. The total resistance takes into account the components linked to bearing, rolling, aerodynamic and additional resistances.

The **ZNet** class, yielding the preceding classes and a series of functions member, consents to obtain a complete and exact description of the network. In fact, in addition to the creation or destruction functions, the
ZNet class has also the control functions on the array of objects of the type Substation, Node, Link and Tab_Res_Moto. Such a control checks at first the dimension of the mentioned array, and then each element of the array. The network solution is assigned to a class derived from ZNet, in fact this last has all the informations for the description of the network in the case more general as possible, for example a.c. case and d.c. case, but the solution algorithms are described in detail in another classes ZNet_DC and ZNet_AC.

In definitive the complete hierarchy of the classes used in the design of the simulator is the following:

One can notice the presence of a root class ZLabel, which consents to label each object in a manner suitable to effect their on-line consultation.

5 Physical Models

In the precedent paragraphs it has been marked the necessity, during the simulator design, to pay particular attention to the models physical aspects. Obviously, only from the knowledge of the exact and detailed analytical models of the various objects it is possible to proceed to the implementation of software codes that succeed to furnish reliable results. In the present case, the physical models which require particular attention, for their wide influence on the results, are those of the motors and of the electrical network (Mellitt, Goodman & Arthurton [7,8], Rambukwella, Mellitt, Goodman & Mouneimne [9], Battistelli, Gagliardi, Losi & Piccolo [10], Battistelli, Carlomusto, Gagliardi & Piccolo [11], Allan, Chan, Mellitt, Anderson & Chiang [12]).

To the purpose to describe the models of the d.c. motors and the methods used for the solution of their equations, it is possible to refer to the more complex case represented by a series excited d.c. motor with starting resistors and field-weakening at constant power, described inside of the
ZMotor_DC_Ind_PC class. For it, the solution of the equations is founded on the algorithm shown in fig. 5.

The weakening degree calculus requires a specific procedure. The motor module checks if the actual weakening degree is such to permit another decrease, comparing this value with the reference one previously given in input. If the actual value overcomes the reference one, the algorithm proceeds executing a second comparison between the reference value and the next value in the weakening sequence. If this new value results compatible with the requests in input, the program proceeds to calculate the corresponding steady state point, in order to obtain the current. This value is compared with one stored and if it results greater the commutation does not take place, otherwise it is proceeded to adopt the new value as the weakening factor.

The exposed resolutive method has the advantage to allow the weakening complete control by the control logic module and not by the motor module, leaving to the motor module the real management of the interested quantities. So doing it is possible to alternate at pleasure phases at full field with other at weakened field (with the possibility to realize intermediary phases with eventually changed configurations of the motors). Once fixed the field-weakening degree, it needs to solve the motor equations.

Relatively, instead, to the solution of the electrical network in terms of voltages and current, it has been considered opportune to take on as unknown quantities the node voltages $v_n$, the trains voltages $v_t$ and the source currents $I_s$. The characterized and implemented solution algorithm foresees the division of the procedures of calculation in two distinct steps. At first, the node and the train voltages are evaluated, only in a second moment the source currents are determined.

Relatively to the first step, a system of equations is built exploiting the method of the nodal voltages. In such a way two groups of equations are
obtained, the first is relative to the network fixed nodes, the second is referred to the trains.

For the automatic construction of the equations relative to the $n_d$ fixed nodes it appears useful to exploit the data organization, chosen for the network representation. In fact, as in each node a table of the links connected with it is present, the coefficients of the equations are obtained by an ordered scan of such a table and by a reading of the trains table. If this results empty, then the conductance of such link will be the total one. If, instead, the trains table contains at least an element, the code proceeds comparing the progressive positions, determining the vehicle nearest to the node in examination and calculating the conductance between the two determined points.

For the equations relative to the $n_t$ trains, the code proceeds directly to the reading of the tables of the trains running on the same track, and then to the individualization of the entities nearest to them (trains or nodes.) The system of so obtained equations can be written in the following matricial form:

$$
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
v_{t,1} \\
v_{t,2} \\
\vdots \\
v_{t,n_t}
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_{n_t}
\end{bmatrix}
$$

The matrix $A$ is a square matrix of order $(n_t \times n_t)$ with not zero diagonal elements and with not zero off diagonal elements only in presence of more trains on one same track; therefore the matrix $A$ results to be a sparse matrix. The vector $b_2$ is formed by constant terms, taking into account eventual substations.

The matrix form of the system equations (1) is linear, whose solution is very easy. It is also interesting to observe that the pivot method results to be particularly quick, in consequence of the matrix structure.

Relatively to the second step for the currents evaluation, the code can proceed only at the end of the first previous step, which has determined the node and train voltages.

The currents evaluations is based on the representation of the substations by an ideal generator of voltage $E$ with a series resistor $R$, and it uses the relation:

$$I_{sk} = (E_k - V_{nj}) \frac{1}{R_k}$$

where the index $k$ is referred to the generic $k$-th substation, and the index $j$ is referred to the generic $j$-th fixed node.
For the knowledge of the source currents will then be sufficient to perform $n_s$ sums and $n_s$ products reducing the computational time.

6 Conclusion

The OOP approach is well fitted for the software simulators of electric network for urban and suburban transportation.

The simulator design is more complex, but the OO modularity permits to simplify the code creation and debugging.

The present paper is a first step in the field of the simulators, and authors future tasks will be the improvement and portability of the code on other types of railway electric network.

The actual code has been implemented on a 486 IBM PC AT machine and an example of the results, referred to voltage and current profiles versus time, is shown in figg. 6,7.

Fig. 6 - Current and voltage in a Substation

Fig. 7 - On board current and voltage
474 Railway Design and Management

References


