SOFTWARE IMPLEMENTATION OF CONTROLLER REPRESENTATION IN THE OVNI SIMULATOR

Luis R. Linares Mazana Armstrong José R. Martí
The University of British Columbia
Vancouver, Canada
llinares@ece.ubc.ca mazanal@ece.ubc.ca jrms@ece.ubc.ca

Abstract – This paper presents the latest advancements in the development of OVNI (Object Virtual Network Integrator) power system simulator regarding the implementation of controllers. OVNI's object oriented description of the power network and its exploitation of the coarse grain parallelization granularity provided by the Multi-Area Thévénin Equivalent (MATE) concept, offers great generality, flexibility and efficiency for implementation in software of controllers. Controller elements (i.e. transfer functions, limiters etc.) are objects of the controller element class, and they interact with each other through a controller element interface. As controller elements are completely encapsulated within their class definition, and their communication interface is clearly defined, the presented approach provides great flexibility in adding new controller elements and controllers to OVNI. A general user interface offers capabilities of custom designing controllers and/or use of predefined ones already available in OVNI.

Keywords: Real-time OVNI simulator, the MATE algorithm, network partitioning of large systems, EMTP solution, controllers, transfer functions

1 INTRODUCTION

This paper presents the implementation of controllers in the UBC's OVNI (Object Virtual Network Integrator) real-time simulator. OVNI is based on the EMTP type discrete-time solution and it was developed in particular for real-time simulation of very large electric networks. OVNI uses the same implicit trapezoidal rule of integration as the EMTP, which is A-stable, and in combination with the Critical Damping Adjustment (CDA) technique [6] for suppression of numerical oscillations, it provides accurate and robust simulation results. The general presentation of the software and hardware concepts behind the OVNI simulator have been described in a number of publications [1], [2], [3], [4], [5].

OVNI is based on the Multi-Area Thévénin Equivalent (MATE) partitioning framework where the network is split into subsystems by the introduction of links. The system of equations describing the interconnected subsystems is a combination of nodal equations of subsystems and branch equations for links [4]. Manipulation of the subsystem matrices reduces the subsystems to their Thévénin equivalents in order to calculate the link currents. When the link currents are known, the subsystem equations can be solved independently. The emphasis needs to be made here that even though the system is partitioned, the overall solution for subsystems and all the network elements (including controllers) is obtained simultaneously without a time step delay.

Advantages of the MATE concept include:

- Computation with smaller, dense subsystem matrices,
- Only the subsystem matrices with topology change need to be recalculated in the next simulation step,
- The nature and individuality of each subsystem is preserved. This allows the use of different integration techniques and/or different integration steps for each subsystem,
- Allows multi-machine hardware solution to achieve real-time computation speeds.

OVNI was conceived and developed as an object oriented application and was implemented in C++. The object oriented approach allows generality and flexibility in adding new power system components. This is especially important for OVNI development, because it enables integration of diverse solutions developed by the team of professors and students at the University of British Columbia. Object orientation of OVNI also allows integration of modules and programs written in other programming languages and developed in other simulation tools such as, for example, MATLAB.

Any power system component in OVNI is an object of a generic element class that provides the interface with the core of OVNI. In the following sections the interface between the power system components and the OVNI core will be described followed by the specifics of the controller implementation.

2 INTERFACING POWER SYSTEM COMPONENTS IN OVNI

2.1 The Element class

One of the design goals in developing OVNI has been to allow the creators of models of different power system components to simply "plug in" their models into the simulation engine (the core). The model developer does not need to learn the details of the core, only the interface between the core and his model. The core's highly efficient and robust design remains protected from well intentioned tampering of model designers in their search for features that favor their particular model, but that may introduce unpredictable side effects for other models. To enable this, a unique general interface has been established, it stands between the core of

the simulation engine and the power system components which in this paper are referred to as *elements*. All elements, included and to be developed, are forms of this abstract class of elements, or explained in terms often used in object oriented programming, they are *objects* which are instantiated from classes that derive their functionality from the abstract class of elements.

The interface between the core and the element models is defined within the abstract element class and includes the following two way communication:

- *Element to the OVNI Core* communicates node identification of connecting *external* nodes at the beginning of the simulation, updated external history sources at every simulation time step and a conductance matrix of the element as seen from its external nodes at a time step following the element's topological change.
- *OVNI Core to an element* communicates the voltages of the element's *external* nodes of connection as produced by the solution of the network.

The interface of an element to the core is graphically depicted in Figure 1. The external nodes of an element are the nodes seen from the outside network. The internal element nodes are reduced or *hidden* from the outside network by employing a technique referred to in this paper as *node hiding* [3].

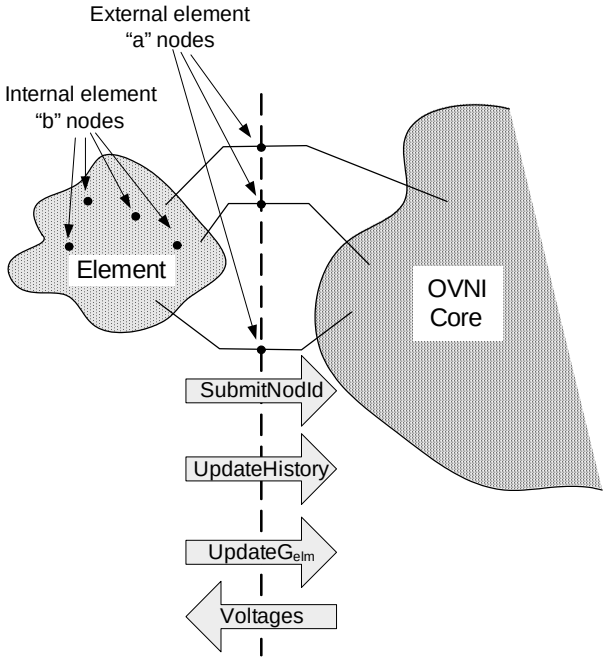

Figure 1: Element-OVNI Core interface.

2.2 Node hiding

Encapsulation of element models brings another question for consideration. Should the outside network be aware of an element's internal structure, or could that structure be somehow hidden away from the outside network when performing the network solution? If the element's internal structure is hidden, the network solution will be obtained faster. However, by doing that, network elements are assigned to perform a task, the one of hiding its internal structure before communicating with the core. Consequently, after receiving external node voltages from the core obtained by the network solution, an element needs to recalculate its internal node voltages in order to update its history terms needed in the next integration step.

This task, however, cannot be performed on a level of the abstract element class, as it depends on the nature of each individual element model. Element models, depending on their topology and characteristics, have the customized code to perform the task of node hiding. Therefore, each element has to be an object of a derived element model class that takes into account its individual topology and characteristics.

Technique referred to as node hiding performs an elimination of internal variables therefore reducing the element to its equivalent as seen from the external nodes. For an element shown in Figure 1, which is composed of discretized equivalent resistances and history sources, system of equations can be written as:

$$\begin{bmatrix} G_{aa} & G_{ab} \\ G_{ba} & G_{bb} \end{bmatrix} \cdot \begin{bmatrix} v_a \\ v_b \end{bmatrix} = \begin{bmatrix} h_a \\ h_b \end{bmatrix} \tag{1}$$

where [G] is the element's conductance matrix, [v] are the element's nodal voltages, and [h] are the element's history source contributions. Subscript "*a*" denotes external node quantities, and subscript "*b*" denotes internal node quantities. By manipulation of matrices, we can express:

$$\left[G_{aa}^{hidden}\right] = \left([G_{aa}] - [G_{ab}][G_{bb}]^{-1}[G_{ba}]\right) \tag{2}$$

$$\left[h_a^{hidden}\right] = \left([h_a] - [G_{ab}][G_{bb}]^{-1}[h_b]\right) \tag{3}$$

The system of equations reduced to the element's external nodes can be written as:

$$\left[G_{aa}^{hidden}\right][v_a] = \left[h_a^{hidden}\right] \tag{4}$$

At every time step the element receives its external node voltages from the core and uses them to calculate internal node voltages needed for updating element's history sources (3).

3 IMPLEMENTATION OF CONTROLLERS IN OVNI

3.1 The Controller Element class

Controllers are generally represented by block diagrams showing interconnections among different controller elements, such as transfer functions, limiters etc. The controller class is derived from the element class, and interfaces with the core in the same way as any other power system component. However, controllers are different than other power system components because their design is not known in advance at the level of the element model class.

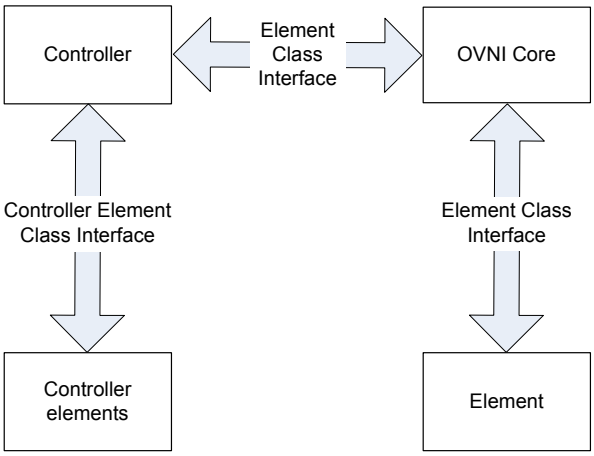

Figure 2: Interfaces in OVNI

An abstract class of controller elements has been introduced to create an interface between controller elements and a controller (Figure 2). Transfer functions, limiters, summers etc. are all members of their respective classes derived from the abstract controller element class.

The two way communication between the controller and its controller elements (the controller element interface) can be described as:

- *Controller element to the controller* communicates identification of its connecting points, updated history terms, and updated coefficients in case when, for example, limiters reach their limits.
- *Controller to controller elements* communicates their input and output values obtained from the network solution needed for updating history terms in the next integration step.

Each controller element submits its coefficients and history to the controller system of equations of a form:

$$[A_{ctrl}]\cdot[x]=[h_{ctrl}] \tag{5}$$

Where $[A_{ctrl}]$ is a matrix of controller coefficients, [x] is a vector of controller variables (inputs + outputs + internal variables) and $[h_{ctrl}]$ is a vector of controller history terms.

Discretized equations of controller elements are obtained, for example, by mapping the Laplace operator *s* to the *z*-domain using bilinear transformation with an integration step Δt:

$$s=\frac{2}{\Delta t}\cdot\frac{1-z^{-1}}{1+z^{-1}} \tag{6}$$

This is described in more details in the following section on the example of a general first order transfer function.

3.2 Disretizition of a first order transfer function

The transfer function block is used to describe a relationship between input $X_1(s)$ and output $X_2(s)$ in the Laplace domain. The first order transfer function is a rational function of the following form:

$$H(s)=K\frac{b_0+b_1 s}{a_0+a_1 s}=\frac{X_2(s)}{X_1(s)} \tag{7}$$

By substitution of equation (6) into (7), a first order difference equation of the following form is obtained:

$$-B_0 x_1(t)+A_0 x_2(t)=h(t) \tag{8}$$

where

$$\begin{aligned} A_0&=a_0+a_1\frac{2}{\Delta t} & A_1&=a_0-a_1\frac{2}{\Delta t}\\ B_0&=K\left(b_0+b_1\frac{2}{\Delta t}\right) & B_1&=K\left(b_0-b_1\frac{2}{\Delta t}\right)\end{aligned} \tag{9}$$

and

$$h(t)=B_1 x_1(t-\Delta t)-A_1 x_2(t-\Delta t) \tag{10}$$

The right side of equation (8) depends on values of input and output from the previous time step, therefore it is referred to as a history term h(t) of a first order transfer function.

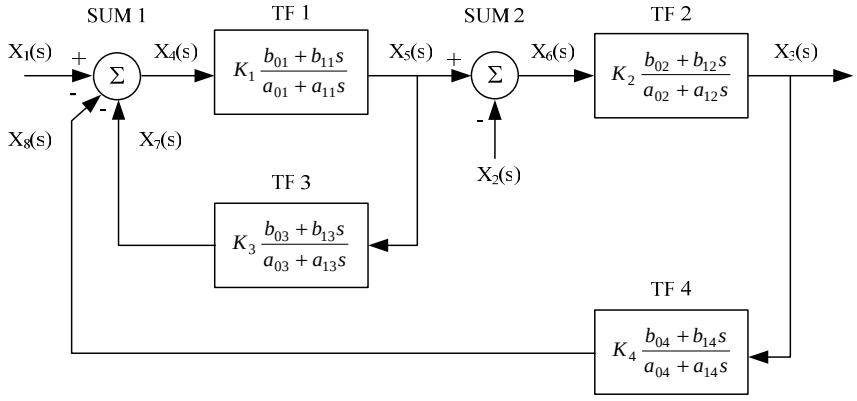

Figure 3: An example of a controller

$$\left[\begin{array}{ccc|ccccc} 1 & 0 & 0 & -1 & 0 & 0 & -1 & -1 \\ \hline 0 & 0 & 0 & -B_{01} & A_{01} & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & A_{02} & 0 & 0 & -B_{02} & 0 & 0 \\ 0 & 0 & 0 & 0 & -B_{03} & 0 & A_{03} & 0 \\ 0 & 0 & -B_{04} & 0 & 0 & 0 & 0 & A_{04} \end{array}\right] \cdot \left[\begin{array}{c} x_1(t) \\ x_2(t) \\ x_3(t) \\ \hline x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \\ x_8(t) \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline h_1(t) \\ 0 \\ h_2(t) \\ h_3(t) \\ h_4(t) \end{array}\right] \tag{11}$$

3.3 System of equations of a controller

As noted before, the controller system of equations has the form of (5). Each controller element contributes with its own equation, for example equation (8) in case of a first order transfer function. Let us consider a controller with four first order transfer function (TF) blocks and two summers (SUM) depicted in Figure 3.

System of equations for this controller with contributions from all six controller elements is shown in (11) where transfer functions' coefficients and history terms are calculated according to (9) and (10).

Controller variables in OVNI are numbered in consecutive order with the controller inputs (x_1, x_2) numbered first, followed by the controller output (x_3), and the internal controller variables (x_4, x_5, x_6, x_7, x_8), as indicated in Figure 3. The procedure equivalent to node hiding is then applied to the system of equations to reduce it to an input-output relationship at every simulation time step.

3.4 Reducing the controller system of equations

Controller matrix [A_{ctrl}] can be partitioned into four matrices as outlined in (11). Controller system of equations (5) can then be written as:

$$\begin{bmatrix} A_{aa} & A_{ab} \\ A_{ba} & A_{bb} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \end{bmatrix} = \begin{bmatrix} h_a \\ h_b \end{bmatrix} \tag{12}$$

where subscript "*a*" denotes external controller variables (inputs and outputs) and subscript "*b*" denotes internal controller variables that are to be eliminated. Equivalent to equations (2)-(4), we can write:

$$\left[A_{aa}^{reduced}\right] = \left(\left[A_{aa}\right] - \left[A_{ab}\right]\left[A_{bb}\right]^{-1}\left[A_{ba}\right]\right) \tag{13}$$

$$\left[h_a^{reduced}\right] = \left(\left[h_a\right] - \left[A_{ab}\right]\left[A_{bb}\right]^{-1}\left[h_b\right]\right) \tag{14}$$

The controller system of equations reduced to external variables can be written as:

$$\left[A_{aa}^{reduced}\right]\left[x_a\right] = \left[h_a^{reduced}\right] \tag{15}$$

As we can see matrix [A_{bb}] has to be invertible in order to reduce the controller system of equations to an input/output relationship.

The first step after constructing controller equations from the input data is ordering of equations to comply with the controller's internal node numbering, as the user writing an input file may choose to order controller elements (and their corresponding equations) in any random order.

An algorithm for the controller equation ordering has been developed to assign each controller element an order number in which they contribute to the system of equations. This sorting of controller elements is done during preprocessing of input data.

By default, controller element order numbering is assigned according to their sequence in the input file. Once all the elements submit their equations, the algorithm checks diagonal values of [A_{bb}], and if any one of them is zero it re-evaluates order numbers of controller elements until the condition is satisfied.

For the case of our controller, for example, controller element order in which they submit their equations is SUM 1, TF 1, SUM 2, TF 2, TF 3 and TF 4.

If, after equation ordering, matrix [A_{bb}] cannot be inverted, it means that the controller system of equations cannot be reduced to an input/output relationship. In such case a controller will contribute its full [A_{ctrl}] matrix to the network equations, and its internal variables will become external. If the solution of the overall system of equations cannot be found once controller equations are included, an error is reported signaling to the user that there is a problem in the controller's design.

3.5 Nonlinear controller elements and MATE

Treatment of non-linear controller elements is essentially equivalent to the treatment of any other non-linear devices in OVNI. The MATE concept introduces link equations into nodal analysis that can conveniently interface nonlinear elements with a linear network.

Let us briefly review the concept of MATE demonstrating properties of link equations. Referring to [2], any system of equations can be partitioned into subsystems by introduction of links. For example, if a system is partitioned into two subsystems, the system of equations has a form of (16).

$$\begin{bmatrix} A & 0 & p \\ 0 & B & q \\ p^t & q^t & -z \end{bmatrix} \begin{bmatrix} v_A \\ v_B \\ i_\alpha \end{bmatrix} = \begin{bmatrix} h_A \\ h_B \\ -v_\alpha \end{bmatrix} \quad (16)$$

where:

- $[A]$ and $[B]$ are admittance matrices of the two subsystems,
- $[p]$ and $[q]$ are current injection (connectivity) matrices,
- $[z]$ is the matrix of links' Thévénin impedances,
- $[h_A]$ and $[h_B]$ are vectors of accumulated currents of the two subsystems,
- $[v_\alpha]$ is the vector of links' Thévénin voltages.

After manipulation with partitioned matrices we obtain the MATE system of equations of the following form:

$$\begin{bmatrix} 1 & 0 & A^{-1}p \\ 0 & 1 & B^{-1}q \\ 0 & 0 & Z_\alpha \end{bmatrix} \begin{bmatrix} v_A \\ v_B \\ i_\alpha \end{bmatrix} = \begin{bmatrix} A^{-1}h_A \\ B^{-1}h_B \\ E_\alpha \end{bmatrix} \quad (17)$$

where:

- $Z_\alpha = p^t A^{-1} p + q^t B^{-1} q + z$,
- $E_\alpha = p^t A^{-1} h_A + q^t B^{-1} h_B + v_\alpha$.

As shown in Equation (17), when solving for link equations, subsystems are reduced to their Thévénin equivalents as seen from linking nodes at every simulation time step. Nonlinear elements are included in the system of link equations and the solution for link currents is obtained. Finally, the subsystems are solved independently of each other with link currents injection representing contributions of other subsystems and nonlinear elements to the solution of the subsystem in question.

3.6 The Controller class

A flow chart in Figure 4 depicts the main functions of the controller class.

Construction of a controller is invoked during preprocessing of input data once the keyword "CTRL" is found in the input file. Preprocessor creates an object of a controller class by calling its default constructor. During the construction of a controller, its data is read from the input file. Controller will receive information on number of external and internal connection points, as well as the nature of the input and output signals. When a keyword for a controller element is found (i.e. "TF1" for a first order transfer function), controller creates an object of a corresponding class (i.e. *tf1_t* class) that reads in data for that particular controller element from the input file. This process is repeated every time a keyword for a controller element is found.

During preprocessing controller matrices and vectors are initialized with the data from the input file. Order numbering of controller elements is performed and the system of controller equations is reduced for submission to the solver.

In the time loop, controller elements are checked for a change in coefficients (for example, a limiter reaches its lower or upper limit) and A_{ctrl} is updated accordingly. Only if the change is detected, the reduction of the matrix will be repeated with updated coefficients. The reduced matrix is submitted to the solver.

Controller history terms are recalculated at every time step. A request is made to each controller element to submit its history terms when provided with the solution for its input/output variables from the previous time step. Once updated, the vector of controller history terms is reduced according to (14) and submitted back to the solver.

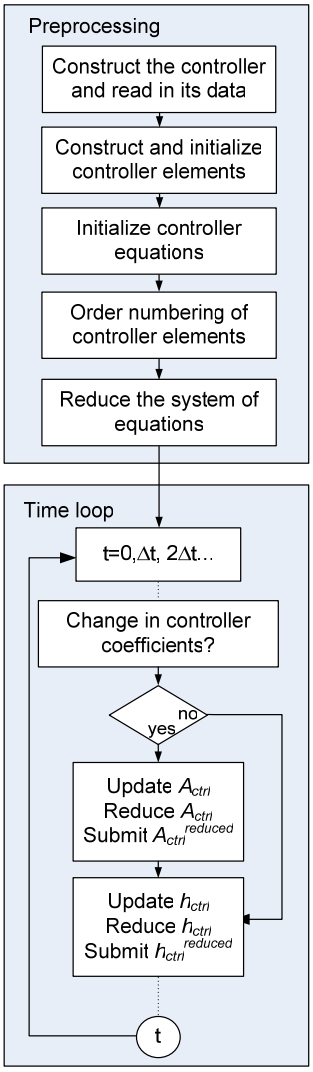

Figure 4: Flow chart for the controller class

3.7 Input format

Input description of controllers in OVNI has a SPICE like format. The .MODEL statement allows the user to specify parameters of controller components and their interconnection points. Input description shown in Figure 5 corresponds to the controller in Figure 3. The user can choose among available controller components to

custom design a controller or use the library of existing predefined power system controller models.

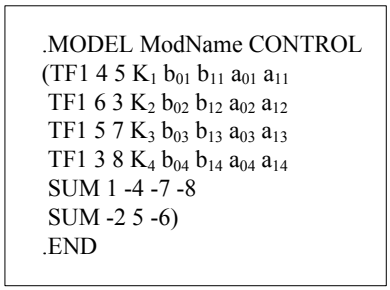

Figure 5: Input format of the controller

3.8 Controllers and the Network solution

As explained in previous sections, controllers submit their coefficients and history terms to the solver for the purpose of obtaining an overall network solution. However, it is the task of the solver to handle controller equations by including them at appropriate locations within systems of equations describing the network links, subsystems, and other network elements. Mainly there are three distinctive types of controllers:

- Type I controller that operates on the network level and it is included in the link equations,
- Type II controller that operates on the subsystem level and it is included in the subsystem equations,
- Type III controller that operates on an element level and it is included in the element equations.

Type I and II controllers are handled within the core of OVNI. Type III controller equations have to be included on the element model level, therefore it is a responsibility of a model developer to deal with incorporation of those controller equations. The model developer has the library of available controller elements at hand to facilitate the controller implementation within their model.

4 CONCUSIONS

OVNI (Object Virtual Network Integrator) is the UBC's power system simulator aimed at attaining real time solutions speeds for analysis of large power system networks using desktop computers. OVNI is based on the MATE concept that allows network partitioning by the introduction of links. The software has been written in C++ using high-level object-oriented design techniques to allow for addition of new modules or the upgrading of existing ones with a minimum of modifications to the core of OVNI.

Models of power system components called *elements* have a unique interface with the core defined within the element class. Elements submit their contributions to the network's conductance matrix following their topological change. They submit their history sources at every simulation time step and in return receive the nodal voltages of their external nodes. The node hiding technique has been used to reduce internal elements structure and increase efficiency in obtaining the network solution.

Controllers have more complex implementation issues than other power system components, one of them is due to their custom design. To make that possible, the class of controller elements has been introduced for the purpose of interfacing controller components with the controller system of equations. From the developer's point of view, this interface enables new controller element models to be simply plugged into OVNI.

Controller description in the OVNI input file has a SPICE like format. Users can choose among available controller components to create their own controller design, and/or use existing controllers available in the controller library of OVNI. Graphical user interface suitable for controller design remains to be developed in the future.

REFERENCES

[1] J. R. Martí, L. R. Linares, "Real-Time EMTP-based transients simulation," IEEE Transactions on Power Systems, Vol. 9, No. 3, pp. 1309-1317, August 1994.

[2] J. R. Martí, L. R. Linares, J. Calviño, H. W. Dommel, J. Lin, "OVNI: An object approach to real-time power system simulators," Proceedings of the 1998 International Conference on Power System Technology, Powercon'98, Beijing, China, pp. 977-981, August 18-21, 1998.

[3] L. R. Linares, "OVNI: (Object Virtual Network Integrator) A New Fast Algorithm for the Simulation of Very Large Electric Networks in Real Time", Ph.D. Thesis, The University of British Columbia, August 2000.

[4] J. R. Martí, L. R. Linares, J. A. Hollman, F. A. Moreira, "OVNI: Integrated Software/Hardware Solution for Real-Time Simulation of Large Power Systems,"Conference Proceedings of the 14th Power Systems Computation Conference,PSCC02, Sevilla, Spain, 7 journal pages, June 24th - 28th, 2002.

[5] J. A. Hollman, J. R. Martí, Real Time Network Simulation with PC-Clusters, IEEE Transactions on Power Systems, Vol. 18, No. 2, pp. 563-569, May 2003.

[6] J. Lin, J. R. Martí, "Implementation of the CDA procedure in the EMTP," IEEE Transactions Power Systems, Vol. 10, No. 5, pp. 394-402, May 1990.

[7] H. W. Dommel, "EMTP Theory Book", 2nd edition, Microtran Power System Analysis Corporation, Vancouver, British Columbia, April 1996

[8] P. Kundur, "Power System Stability and Control", McGraw Hill, 1994

[9] J. G. Proakis, D. G. Manolakis, "Digital Signal Processing", Third edition, Prentice Hall, 1996