Modeling and Simulation for Relay Protection with the CD++ Toolkit

Hong-Shan Zhao, Ji-Ping Zhang and Zeng-Qiang Mi

Abstract--**This paper presents a new method of modeling and simulation for relay protection by using a DEVS-based toolkit named CD++. With this toolkit, the modeling of relay will be more easy and straightforward. We takes the digital three zones distances protection as the example, studies the approach of ideal protection relay modeling, and also validate the validity and feasibility of the model through analysis.**

Keywords--**CD++, DEVS, modeling, relay protection.**

I. Introduction

Research of modeling and simulation for relay protection is in favor of not only understanding relay's dynamic behavior as disturbance and faults happens in power system, but also reducing the development times and costs of new products of system[1]. Through modeling and simulation, we can aware the mechanism and process of continuous faults led by behavior of relay during power blackout in power system [2]. In electrical power system, the tools used for studying on modeling and simulation to the digital relay protection, such as EMTP, ATP or MATLAB and so on, are mostly all not very convenient, comprehensible, and especially more complex regarding the complex logical relation modeling approach. So, it is necessary to study a new method to improve the modeling of relay protection.

The protection relay appear as the discrete event dynamic behavior, that is, logic dynamic relation, while the CD++ toolkit is adaptable to Discrete-Event modeling and simulation[3]. So we present a thought and steps of modeling and simulation for relay protection in CD++.

II. Basic Knowledge of Modeling for Relay

In power system, relay protection plays important role. Its behavior will affect the reliability and safety of the whole system. Therefore, it is necessary to construct a relatively accurate and real model for relays. Through analysis, the ideal distance relay dynamic behavior may be abstracted as three operational modes, namely, normal state (*Norm*), state after startup component action (*StartPost*) and state after operation (*OperPost*). There are strict and fixed relations between each two modes. Fig. 1 shows their relations: under the Norm, there are no any faults in system. Relay will do nothing at this mode; If fault appears, relay's three zone (**I, II** and **III**) protection will go to the mode *StartPost*. According to every zone's optional mechanism, they will decide which mode to enter.

A. The Relation of Three Modes of Relay

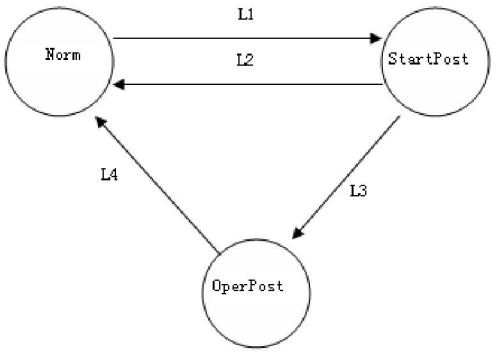

Fig. 1 The relation of three modes of relay behavior

For every zone of relay protection, the arcs L1, L2, L3 and L4 have their own meaning respectively. For example, for **zone I**, L1 means that fault happens somewhere in itself line (the line equipped with the relay). And now, **zone I** must startup and enter into mode StartPost. If the fault is located in the precinct of **zone I**, then it will enter into OperPost through L1; If the fault appears at the end of this line, that is, out of the domain of zone I, the relay of **zone I** will be back to Norm. While for **zone II**, L1 means that fault happens in its line or part of neighbor line. If this condition happens, **zone II** will enter into StartPost. Then, it will go to OperPost though L3 if the fault happened at the end of this line. Therefore, every zone's arcs described in Fig. 1 have different meaning.

B. DEVS Model of Relay in CD++

To achieve these logic relations, as mentioned advance, we will model the behavior of relay protection with CD++ toolkit. CD++ is a tool for Discrete-Event modeling and simulation, based on the DEVS (Discrete EVents Systems specifications) formalism. It runs either in standalone (single CPU) or parallel mode (over a network of machines). CD++ also provides a graph-based definition of DEVS models for more straightforward way to define models. Graph-based notations have the advantage of allowing the modeler to think about the problem in a more abstract way. Therefore, we use the extended graphical notation to define the models of relay.

H. S. Zhao is with Dept. of Electrical Engineering, North China Electric Power University, China (zhaohshcn@126.com)

J. P. Zhang is with Dept. of Electrical Engineering, North China Electric Power University, China (aendey@163.com)

Z. Q. Mi is with Dept. of Electrical Engineering, North China Electric Power University, China (mizq@ncepu.edu.cn)

As a DEVS-based tool, models must be defined through DEVS theory system. In DEVS, models are divided into basic model (Atomic Model) and more complex model (Coupled Model). A Coupled model usually includes atomic models or other coupled models. That is, DEVS allows modular description of models that can be integrated using a hierarchical approach [4].

A DEVS atomic model is formally described by:

$$M =< X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, D >$$

Here, X is the input events set, S is the state set, and Y is the output events set. There are also several functions: δ_{int} manages internal transitions, δ_{ext} external transitions, λ is the outputs, and D the elapsed time.

A DEVS coupled model is defined as:

$$CM =< I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$$

Here, X is the set of input events, and Y is the set of output events. D is an index of components, and for each $i \in D$, M_i is a basic DEVS model, where

$$M_i =< I_i, X_i, S_i, Y_i, \delta_{\text{int}_i}, \delta_{ext_i}, tai >$$

I_i is the set of influences of model i. For each $j \in I_i$, Z_{ij} is the translation function from i to j.

CD++ implements the DEVS theory. Each DEVS graph defines the state changes according to internal and external transition functions, and each is translated into an analytical definition. In CD++, DEVS graphs can be formally defined as:

$$GGAD =< X_M, S, Y_M, \delta_{\text{int}}, \delta_{ext}, \lambda, D >$$

$X_M = \{(p, v) \mid p \in IPorts, v \in X_p\}$ set of input ports;

$Y_M = \{(p, v) \mid p \in OPorts, v \in Y_p\}$ set of output ports;

$S = B \times P(V)$ states of the model;

$B = \{b \mid b \in Bubbles\}$ set of model states.

$V = \{(v, n) \mid v \in Variables, n \in R_0\}$ intermediate state variables of the model and their values.

δ_{int}, δ_{ext}, λ, and D have the same meaning as in traditional DEVS models [5].

Therefore, we can define a relay models from the following way: First, define atomic models for every zone of distance relay protection. In atomic model, we use a state (Bubble) to represent a mode. So, the relay has three states (Norm, StartPost and OperPost). The time interval of every state and input/output port will be defined. Second, according to strict logic relation showed in Fig.1, we can define all the internal/external transition functions to connect the three states. And then, the fourth atomic model named Break could be defined for more clear observation of relay's behavior. Break model's output information will tell us which zone sent the "open" signal to breaker. Finally, we will integrate four atomic models into a coupled model through logical connections.

III. CD++ MEDELS OF DISTANCE RELAY PROTECTION

We will use a graphic interface named GGAD of CD++ to define the models of relay. In atomic model of GGAD, when certain event is received form an input port, the model will enter into another state by calling internal or external transition function. Every state has its default interval to remain the current state. If the received information satisfies the condition of external transition function, the model will enter into corresponding state immediately before the interval of current state using up. However, if conditions of all external transition function are not satisfied, the model will first send a value to an output port, and then go to other state after the default interval exhausted. The following is the steps of modeling for three zone distance relay protection.

A. Atomic Models of Zone I/II /III and Break

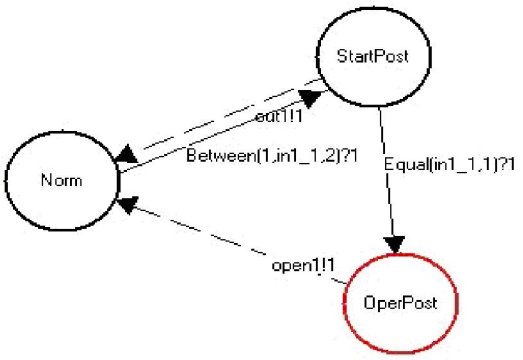

Fig. 2 Atomic model of **zone I**

We define that the initial state of **zone I** is *Norm*. In Fig.2, there is one input port named *in1_1* and two output port: *out1* and *open1*. The fault type can be identified form the event information received from *in1_1*. The possible value of the input port could be 0, 1 and 2. The value 0 is defined as no fault happens; 1 and 2 are defined as fault happens within the domain of **zone I**, and out of **zone 1**, that is, a fault appeared at the end of the line. We can get the fact from Fig. 2 that **zone I** will startup and enter into *StartPost* if the value from *in1_1* is 1 or 2 (the condition of external transition function *Between (1,in1_1,1)?1* is satisfied), the function *Equal (in1_1, 1)?1* is used so that the model to enter into state of *OperPost*. After a short interval, **zone I** will send 1 (defined as a command of line release) to output port *open1*, and then will be back to state of *Norm*.

Fig.3~5 shows the models of **zone II**, **III** and **Break** in GGAD.

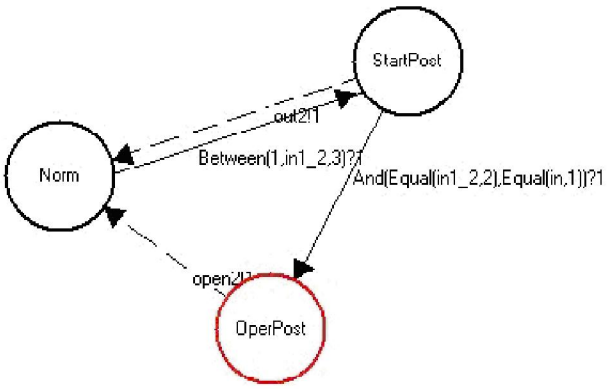

Fig.3 Atomic model of **zone II**

The meaning of functions in **zone II** is similar to **zone I**. There are two input ports (*in1_2, in*) and two output ports (*out2, open2*) in this model. Port *in1_2* still describes the type of fault. Its value is 0, 1, 2 and 3. Here, the meaning of 0~2 is as same as *in1_1* of **zone I**. While 3 indicates that fault happened at the neighbor line, and the location of the fault is still in the precinct of **zone II**. Port *in* is used for identification of sate of **zone I**. It will connect with the output port of **zone I** (that will be discussed later). If the value of *in* is 1 and *in1_2* is 2 (*And (Equal(in1_2, 2),Equal(in, 1))?1*), that means **zone I** hasn't entered into *OperPost* through *StartPost,* there is a fault at the end of the line. So **zone II** operates and enters into *OperPost*. And after a short interval, it will send value 1 (the same meaning as *open1*) to output port *open2* and then go back to *Norm*.

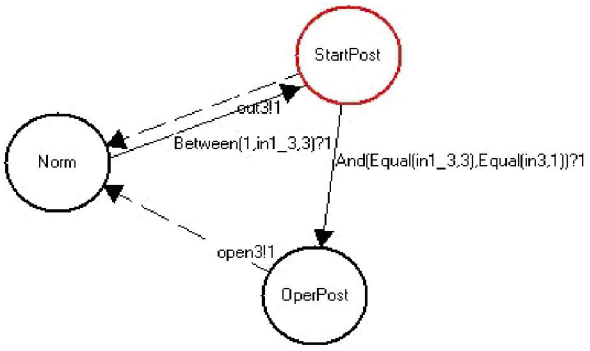

Fig. 4 Atomic model of **zone III**

Form Fig. 4, we notice that two input port is defined in **zone III**. The values of port *in1_3* are 0~3. The meaning of 0~2 is as same as **zone II**, while 3 is different. When 3 is received form *in1_3*, which means a fault out of domain of **zone II**. The port *in3* will connect with *out2*. So when the condition of the external transition function (**And (Equal(in1_3, 3),Equal(in3, 1))?1**) is satisfied, **zone III** will operate and enter into *OperPost*. There should be a short delay before it enters into *OperPost* from *StartPost* when fault appeared. The delay will be expressed through the interval of *StartPost* in **zone II** (that will be discussed later). The output port *open3* has the same meaning of *open1/open2*.

The last atomic model is show in Fig. 5

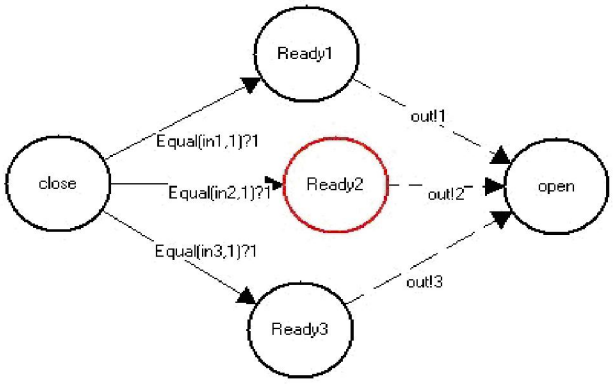

Fig. 5 Atomic model of **Break**

In order to analysis the behavior of relay, we have designed three basic states for **Break**: *close, ready, open.* Their respective meanings are state of *close* of switch, state of ready to break the switch, state of *open* of switch. Here, *ready1~ ready3* used to represent receiving signals form **zone I**, **II** and **III** respectively. Three input port (*in1, in2* and *in3*) receive signals from three zone's output port. The model **Break** has an output port *out*. Its value could be 1, 2 and 3 (represent respectively that switch opened by receiving *open* signal form **zone I**, **II** and **III**). The three states of the atomic model **Break** is presented to illuminate the relation of three zone and switch.

B. Coupled Models of Relay

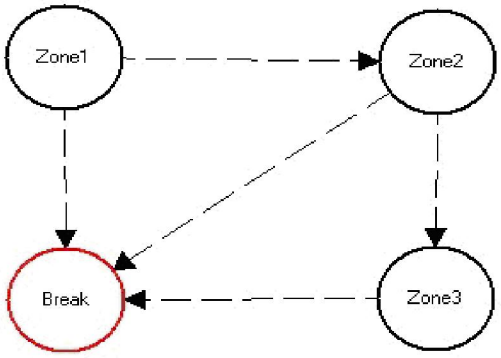

Fig. 5 Coupled model of relay protection

When fault happens, the three zone of relay will receive corresponding information at the same time. So there should be three input port in coupled model. The three input port of coupled model are: port1, port2 and port3. They will connect with the input port *in1_1, in1_2* and *in1_3* respectively. An output port *out* will connect with the output port of **Break**. To connect all atomic models, we need to use the following rule: an output port of source model must connect with one of the inputs port destination model; an input/output port of coupled model must connect with one of the input/output ports of an atomic model. For example, the connection *Zone1.out1-Zone2.in* and *MODEL.port1-Zone1.in1_1* follow this rule. All the connections of coupled model of relay protection are shown in Fig. 6.

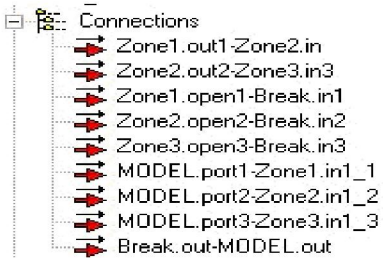

Fig. 6 connections of couple model

From this way, we get the whole module of relay protection. As mentioned above, the three input ports will inform every zone with the type of faults, the output port can give us two kind of information: 1) whether the switch has open or not; and 2) which zone sent the *open* signal to switch.

IV. VALIDATING OF RELAY MODEL

Rationality of the model of relay is analyzed through the following two types of faults (which occurred in the line equipped with the relay, other line's fault will not be considered):

(1). Fault occurred in the domain of **zone I**

Under normal condition, the values of input port and output port in protection module are zero. When fault (1) occurring, protection module will receive fault information from three input ports and output ports. This time, the values of three ports is: port1 (*in1_1*) =*port2* (*in1_2*) =*port3* (*in1_3*) =1. When satisfying condition of external transition function *Between ()* in every zone, the **zone I**, **II** and **III** will operate and enter into *StarPost* state. For **zone I** satisfying the condition *Equal (in1_1, 1)*, **zone I** will operate and enter into *OperPost* state. After 5ms time delay, it returns to *Norm* by internal transition function *opten1!1*. Now, for *open1* connecting port *in1* of Break module, Break enters into Ready1 state and enters into open with 30ms time delay after receiving the port's value 1 before sending *out*=1 to output port, that indicates the open signal to switch was received from **zone I**. Therefore, it will need 35ms time from finding fault to remove fault regarding **zone I**. It needs to judge the value of port *in (out1)* when **zone I** and **zone II** enter into *StartPost* simultaneously. At that time, **zone II** does not sent 1 to port *out1* and does not operate because **zone I** has entered *OperPost* already. At the same time, **zone II** sent 1 to *out2* and return. After zone III starting, it will not operate for dissatisfying the condition *in1_3*=3.

(2). Fault occurred at the end of the line.

Every zone will startup at the same time and enters into *StartPost* when this fault occurred. **Zone I** will not operate and return to Norm with a delay (500ms) for dissatisfying its operation condition. During the delay, **zone II** will not enter into any other states and keep waiting in the state StartPost for a longer interval (>500ms) until receive the value 1 from out1 of **zone I** after 500ms. Now, **zone I** is returning to Norm, while **zone II** will enter into *OperPost* for the function ***And (Equal(in1_2, 2),Equal(in, 1))?1*** is satisfied. As same as **zone I**, an open signal to switch will be sent out to port *in2* of **Break**. The time is 535ms from the fault was detected to removing it. **Zone III** will not operate for not receiving any information from *out2*.

Similar analysis can be done for other types of faults.

CD++ provides a platform for simulation. Each simulation of models of relay protection was executed. We can validate the reasonability of all models we built from the results of execution.

V. CONCLUSION

This paper presents a new method for modeling and simulation for ideal relay protection in power system. The detailed process of modeling is provided with a straightforward way. With the CD++ toolkit, we also can define more strict and precise models for relay. There are still some shortages in modeling. For example, we did not take into account the instance when oscillation of power system happens, but we are working at it now.

VI. REFERENCES

[1] MI Zeng-qiang, ZHAO Hong-shan, and WANG Hai-ping, "Abstract Modeling Method of Digital Protection Relay," *Proceedings of the CSEE*, vol. 25, pp. 51-56, Nov. 2005.

[2] Zhao Hongshan, Mi zengqiang, Niu Dongxiao et al. "Power system Modeling Using Hybrid system theory" *Proceedings of the CSEE*, vol. 23, pp. 20-25. 2003.

[3] WAINER, G."CD++: a toolkit to define disrete-event model". G. Wainer. Software, Practice and Experience. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002.

[4] Hong, G. P., T. G. Kim. "A framework for verifying discrete event models within a DEVS-based system development methodology." *Transactions of the Society for Computer Simulation*, Vol. 13(1) pp. 19-34. 1996.

[5] Gastón Christen, Alejandro Dobniewski, Gabriel Wainer. "Modeling State-Based DEVS Models in CD++," Military Government and Aerospace Simulation Symposium. ISBN: 1-56555-279-2. pp. 105-110.