A Co-Simulation-and-Test Method for CAN Bus System

Chu Liu and Feng Luo
Clean Energy Automotive Engineering Center, School of Automotive Study, Tongji University, Shanghai 201804, China
Email: liuchu1985@126.com; luo_feng@tongji.edu.cn

Abstract—Simulation and test are important for the development and validation of automotive ECU (Electronic Control Unit). However, problems arise in the combination of today's automotive control algorithm development and function test. There are lots of professional tools for simulation and test, but lack of interfacing capability between them, especially in the simulation and test of today's automotive network. The approach of COM (Component Object Model) technology is used to solve this problem, and a complete co-simulation-and-test system is introduced in this article, which is made up of CAN (Controller Area Network) Bus Simulation Environment – AutoCAN, and CAN Bus Disturbance Generator – CANspider and MATLAB. AutoCAN and CANspider have great advantages in simulation and test of automotive CAN bus, while MATLAB has superior performance in modeling. The integration of these tools makes it possible to realize the internal communication between them, and combine their respective advantages and increase the efficiency of engineering application. Experiments are performed to prove the effectiveness of the platform for developing and testing the automotive network system based on automotive CAN Bus.

Index Terms—AutoCAN, CANspider, MATLAB, COM, Co-simulation-and-test

I. Introduction

With the rapid development of automotive electronic technology, the communication network on-board is becoming more and more complicated, including the development of the control algorithm as well as the validation of the functionality and reliability.

MATLAB is a high level programming language and interactive environment developed by MathWorks. It is used for algorithms development, data visualization and analysis, numerical computing and interfacing with programs written in other languages to share the resource. MATLAB/Simulink plays an important role in the system level modeling and validation of the automotive product [1], however, without specific toolboxes provided by the tool vendor, it has limited ability to access the automotive CAN Bus and perform real-time simulation and validation.

IHR's AutoCAN is a powerful tool for automotive CAN bus design, simulation and test, which is able to simulation a whole CAN network, create communication among virtual nodes and real nodes on the CAN bus. Various measurement windows make signal displayed and monitored much easier, and its built-in AC Language provides unlimited possibilities for simulation and test. Using its unique AC programming language, the behavior of virtual CAN node can be controlled with ease, however, the ability of its AC language is limited in developing algorithms in the increasingly complex CAN bus system.

IHR's CANspider is a CAN Disturbance Generator, it provides a high-performance CAN test environment, which can be used not only for testing the given CAN nodes, but also the specific CAN network system. With CANspider the user can intentionally and reproducibly disturb the CAN bus. With CANspider, the fault recovery ability of the automotive ECU can be fully tested. CANspider can be easily interfaced with other applications.

In order to reduce the complexity, the method of co-simulation-and-test of CAN Bus with AutoCAN, CANspider and MATLAB is necessary [2], so as to combine with their respective advantages and increase the efficiency of engineering applications. Thus MATLAB can also participate in the communication on CAN bus, and make the data on CAN and in MATLAB associated and synchronous, while simulating disturbances that may occur during the actual working process, and evaluate the real-time performance and reliability of the system by test.

Nowadays, there are some relevant applications in the field of CAN bus system on-board, such as Dual Clutch Transmission (DCT) control system and Electronic Throttle Control System (ETCS), by using of CANoe and MATLAB/Simulink [3]-[4]. It can be seen that the integration of CAN bus simulation tool and modeling tool makes the simulation-and-test platform much more efficient and complete, contributing to the development of automotive CAN bus system.

In this paper, the method of creating communication between different applications via COM technology on Windows is introduced [5]. A test system is built based on this method [6]-[8], which tests the algorithm of the ABS (Anti-lock Brake System) system in the MATLAB, simulates the whole CAN network communication in AutoCAN. Unlike other co-simulation-and-test solutions, CAN bus disturbance is also injected in the network by controlling the CANspider, so as to test the fault tolerance and fault recovery capabilities of a specific control algorithm. A stand-alone test engine is built to

Manuscript received August 16, 2013; revised October 29, 2013.
This work was supported by National High-tech R&D Program of China (863 Program) under Grant No. 2011AA11A214.
Corresponding author email: liuchu1985@126.com
doi:10.12720/jcm.8.10.681-689

execute a list of test cases. This gives the developer and test engineer access to all the test tools in one test bench simultaneously. With the help of the interfacing technology [9], complex simulation and test problems can be solved with high efficiency.

II. Research Method

A. System Structure Definition

The system is mainly composed of four parts: AutoCAN, CANspider, MATLAB and the interfaces between MATLAB and the real CAN bus as shown in Fig. 1.

The AutoCAN hardware is connected through a USB cable from PC (Personal Computer) to the real CAN bus, CANspider can be connected to the CAN bus in the same way, however, it is usually connected in series in the CAN network, so as to achieve programmable break and short circuit on the CAN bus.

MATLAB is a stand-alone software platform, which connects to the CAN bus through the interfaces provided by AutoCAN and CANspider.

During the simulation process of the system, all messages transmitted on the CAN bus by virtual or real CAN nodes can be monitored by MATLAB, or disturbed by MATLAB applications through the interfaces provided. The validation and test of the ECU can be better achieved based on the system setup.

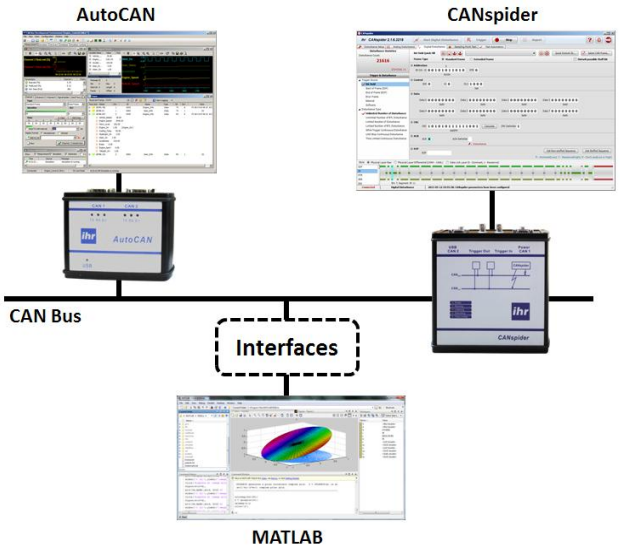

Figure 1. The definition of the system structure.

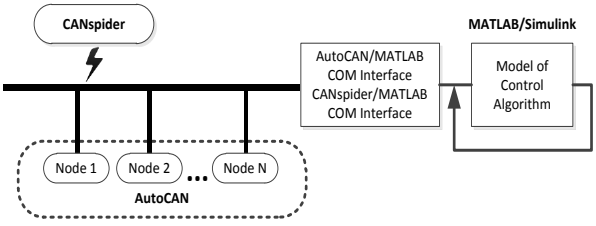

Figure 2. The interfacing principle of the system.

B. Interfacing Principle of the System

Fig. 2 illustrates the interfacing principle among multiple tools of the co-simulation-and-test system.

The seamless integration of AutoCAN, CANspider and MATLAB is based on OLE (Object Linking and Embedding) Automation that allows one application (the client or controller) to control objects exported by another application (the server). The automation client is able to access the objects, properties and methods in the automation server, which complements each other's advantages. The core element of automation is Component Object Model (COM), which defines a group of APIs (Application Programming Interfaces) and binary standard to enable the component objects, from different platforms and in different languages, to communicate with each other well [10]. Since the COM technology is widely used in the automation all over the world, and MATLAB has COM support, it is necessary that this interfacing technique is studied.

The communication between the automation server and client is on the basis of client request and server response. Specific functions as well as the interface for accessing these functions are defined in the automation server. Upon receiving the request submitted by the automation client, the corresponding events are triggered and processed in the automation server, the results are finally sent back to the client. In Fig. 2, AutoCAN and CANspider act as automation servers, while MATLAB/Simulink acts as an automation client. After the communication is established between the client and servers, MATLAB has the ability to transmit or receive CAN messages through the AutoCAN hardware on the bus, and also disturb the specific CAN identifiers through the CANspider hardware on the CAN bus. In this situation the AutoCAN and CANspider can be understood as communication interfaces while MATLAB is the controller.

C. Co-Simulation-and-Test Method

The Co-simulation is composed of the network simulation in AutoCAN and control algorithm simulation in MATLAB/Simulink. The main control logic of the co-simulation resides in the M-script of MATLAB. The behavior of the virtual node in the AutoCAN simulation environment is influenced by the M-script.

The test is also carried out by the M-script in the MATLAB, which includes the signal modification and error injection of the CAN network.

III. The Implementation of the Simulation-and-Test System

A. The Creation of Automation Servers

To create the automation objects and establish the communication, the MATLAB script file (also known as M-File) is adopted because it is the most effective method for executing a series of MATLAB instructions without typing all of the commands from the keyboard. To create

the COM server, the following MATLAB function is used:

h = actxserver('progid');

The function creates a local OLE Automation server, where progid is the programmatic identifier of an OLE-compliant COM server, and h is the handle of the server's default interface. Since the programmatic identifiers of AutoCAN and CANspider have been already registered in the system after the software installation and can be found in the software manual, the following statements create the server objects:

```
srvAutoCANMeasurement =
actxserver('AutoCAN.Measurement'); // Creates the
Measurement Object of AutoCAN
srvAutoCANSimulation =
actxserver('AutoCAN.Simulation'); // Creates the
Simulation Object of AutoCAN
srvCANspider =
actxserver('CANspider.CANDisturbance'); // Creates the
Disturbance Object of CANspider
```

There are multiple interfaces in AutoCAN, including "Measurement", "Simulation", "Hardware" and so on. Each interface has a subset of relevant functions. To use a function under the specific interface, the corresponding automation object should first be created.

TABLE I: FUNCTIONS DEFINED BY IMEASUREMENT INTERFACE OF AUTOCAN

Function Prototypes	Descriptions
int32 TX(handle, int32, int32, int32, int32, int32, int64_T)	Transmits CAN messages
[int32, int64_T, Variant(Pointer)] GetTrace(handle)	Receives CAN messages
void ClearData(handle, int32)	Clears the data display
bool ConnectAutoCAN(handle, bool)	Connects AutoCAN hardware

TABLE II: FUNCTIONS DEFINED BY IDISTURBANCE INTERFACE OF CANSPIDER

Function Prototypes	Descriptions
int32 CANspider_Connect_Hardware(handle)	Connects the CANspider hardware
[int32, char, uint32_T, uint32_T] CANspider_Get_Disturbance_Counter(handle)	Gets the real-time disturbances counters
int32 CANspider_Config_Digital_Trigger_Source_SOF(handle)	Configures the start of frame trigger in digital disturbance
int32 CANspider_Start_Disturbance(handle, int32, string)	Executes the analog or digital disturbance

After the creation of the server objects, the hardware of AutoCAN and CANspider can be accessed using the interface functions of the server objects.

To acquire the list of the supported interfaces, the interfaces function is used to return a list of all the available interfaces such as "srvAutoCAN.interfaces;".

To get the list of all the supported functions of the specific server object interface, the following command is applied:

invoke(srvAutoCAN);

After the execution of the invoke function, the available function is listed by MATLAB. Table I and Table II shows some important functions published by AutoCAN automation server and CANspider automation server.

B. Access to the Functions in the Automation Server

With the automation objects created in MATLAB, all the supported functions can be executed in the following manner "Object.Method", for example:

status = srvCANspider.CANspider_Connect_Hardware;

Where the "srvCANspider" is the automation server object and "CANspider_Connect_Hardware" is its method. After the function call, the variable "Status" can be used to determine whether this function is successfully executed during the simulation or test process. All the methods of AutoCAN and CANspider can be accessed in this way.

There is a difference between AutoCAN automation server and CANspider automation server. The AutoCAN automation server is out-of-process COM server, which means the whole AutoCAN GUI interface will show up after the automation server object is created, thus the measurement data and simulation process can be monitored while the server object is being controlled. The CANspider automation server is in-process COM server, the software GUI interface will not show up during the function call to the automation server object. So the client application has to call the automation server's function CANspider_Get_Disturbance_Counter in real-time to get the results returned from disturbance execution.

Using M-script to interact with the automation server is simple and straightforward. The following M-script demonstrates the usage of AutoCAN automation server, which brings up the AutoCAN GUI, sends a frame on channel 1, and then close AutoCAN:

```
function AutoCAN_TX
he = actxserver('AutoCAN.Measurement');
he.WaitUntilAutoCANIsUp
status = he.TX(1, 258, 1, 1, 8, hex2dec('FFAA55'));
if 1 == status
  disp('Message 0x102 is transmitted');
else
  disp('Message 0x102 transmit failed');
end
he.delete;
disp('autocan closed');
```

C. Interactions with the MATLAB GUI

To achieve better control over the co-simulation-and-test process, it is convenient to create GUI (Graphical

User Interface) in MATLAB. Each GUI interface has its own handles data structure, the GUI components inside can be accessed by this data structure.

Fig. 3 shows the CANspider disturbance module implemented in MATLAB GUI, which is able to configure the CANspider hardware during the simulation, to disturb specific ECUs.

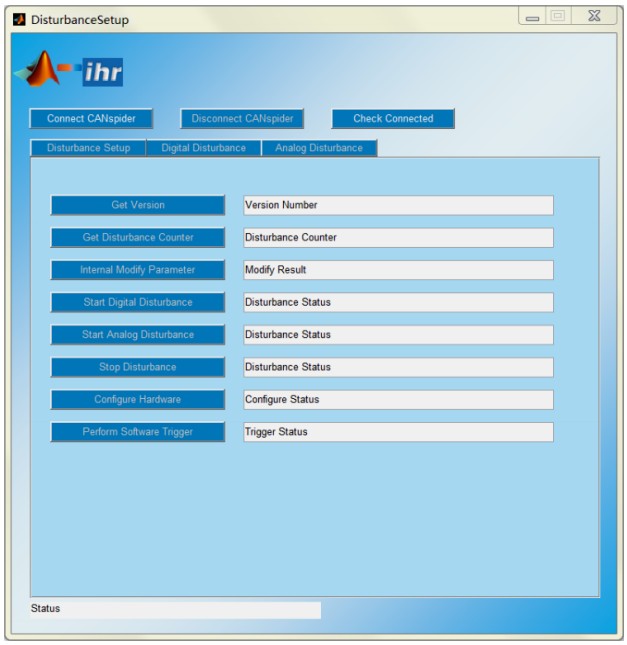

Figure 3. CANspider disturbance module implemented in Matlab.

Take a button component named "Connect CANspider" in GUI as an example, which is used to connect the CANspider hardware, the command can be added into its call-back function in the following manner:

set(handles.ConnectCANspider, 'Enable', 'off');

When this button is pressed, the corresponding call-back function is executed. The Enable property of this component can be on, off or inactive, while this command sets the Enable property to off, to prevent this button from being pressed twice after the successful connection to the CANspider hardware.

Based on the MATLAB GUI, the system is built with Simulink. To ensure the communication between Simulink and MATLAB GUI, the following Simulink operation functions are used:

- Sim – starts the simulation
- Simset– sets the parameters of the simulationsimget – gets the parameters of the simulation
- evalin – to access the variables in the workspace in the M file
- assignin – to copy the variable values from the M file to the workspace

The internal data transmission between MATLAB GUI and Simulink mainly includes two parts, one is the signal input and the other is the output of the simulation result.

To input signal from MATLAB GUI into the Simulink environment, the signal value can first be stored into a local variable, and then transferred into the workspace using the command assignin with a new variable name; the "From Workspace" module of the Simulink is able to retrieve the signal value by the variable name specified in the workspace. The simulation results returned by the simulation of the Simulink model can also be retrieved in the same manner using evalin function. Fig. 4 shows the main control interface of the system built with MATLAB GUI, from which the Simulink model can be invoked and executed.

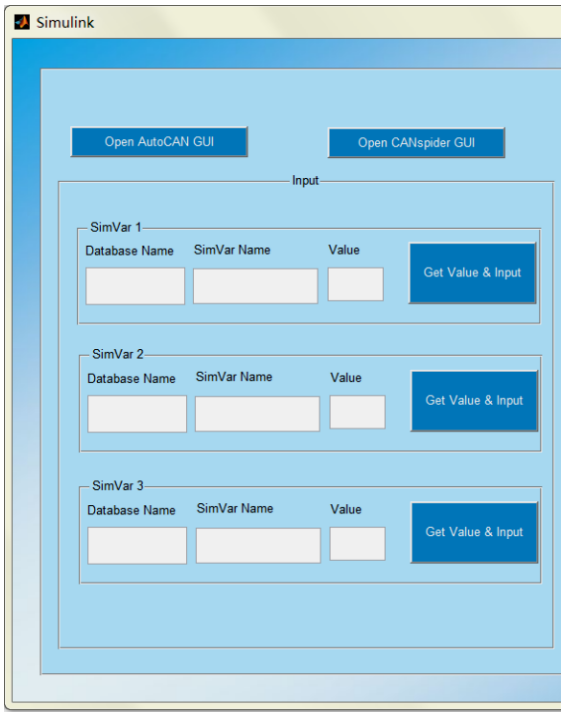

Figure 4. The main control interface built with MATLAB GUI.

D. *System Modeling of Algorithm and CAN network*

The realization of the system modeling can be divided into three parts: the control algorithm model in MATLAB/Simulink, the network model in AutoCAN and the communication between AutoCAN and MATLAB/Simulink.

To achieve hardware-in-the-loop simulation, a Simulink model has to be adjusted to satisfy the COM interface. The adjustments include the conversion of signal input-output components and environment variable input-output components into the "From Workspace" and "Display" modules, the input and output behaviors are controlled by the M script, so as to realize the connection between AutoCAN and MATLAB/Simulink.

Fig. 5 shows an example of the control algorithm model of automotive ABS sub system, in which the elliptical annotations represent the input and output modules that are adjusted for the COM communication.

The CAN network model is realized in AutoCAN, as shown in Fig. 6, which includes the simulated nodes such as "ABS", "Meter" and a tester node.

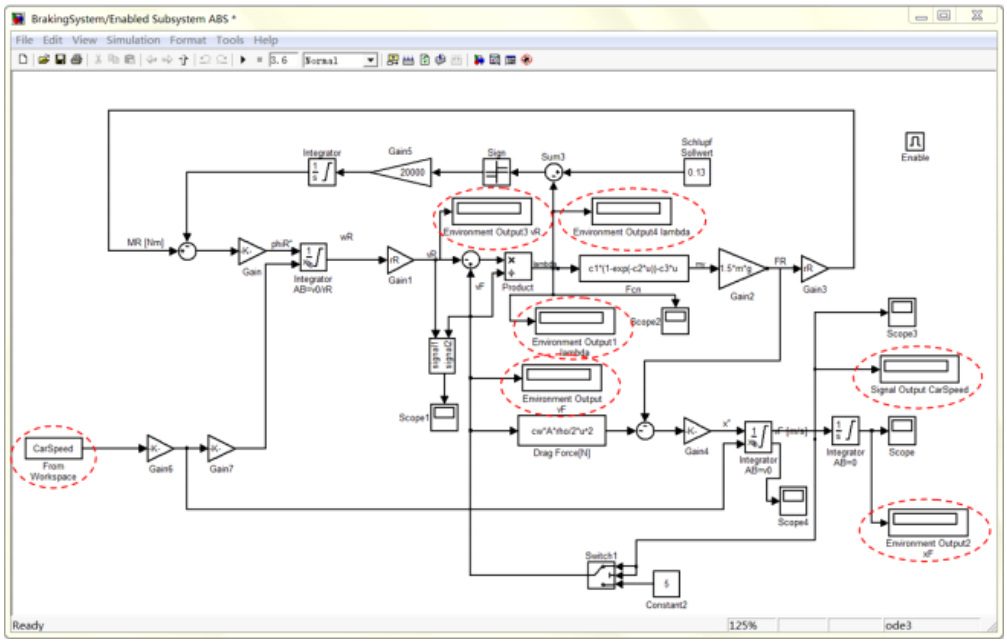

Figure 5. The control algorithm model of automotive ABS system.

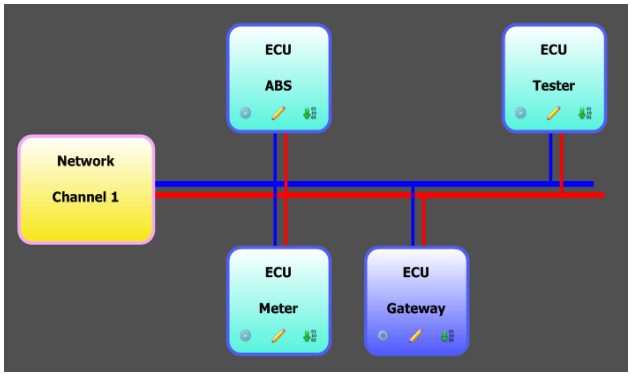

Figure 6. The realization of the network model in AutoCAN.

CAN bus databases are imported into this network for the symbol based CAN signal display and analysis. Each virtual node in the network has been interconnected with the corresponding AC simulation scripts so as to control the data exchange over the CAN network.

The communication between AutoCAN and MATLAB/Simulink is implemented using M scripts by the following methods:

(1) MATLAB/Simulink read/write the internal variables in the virtual node, which uses the function GetSimVarValue published by the interface ISimulation of AutoCAN.

[k, d] = srvAutoCANSim.GetSimVarValue('ABS', 'simBrakeActive');

The input parameter "ABS" specifies the database name of the current simulation system, while the parameter "simBrakeActive" is the internal simulation variable in the AC runtime environment of AutoCAN.

The return value k represents whether the function successfully executed, and the value d represents the real-time value of the simulation variable "simBrakeActive" in AutoCAN.

(2) MATLAB/Simulink sends CAN messages through AutoCAN, which is achieved with the TX function published by the interface IMeasurement of AutoCAN.

retn = srvAutoCAN.TX(1, hex2dec('50'), STD_FRAME, DATA_FRAME, 2, output_speed);

The first input parameter "1" represents the usage of channel 1 of AutoCAN hardware; and the second parameter is the identifier of the message being transmitted; the third parameter "STD_FRAME" represents standard frame type; and the fourth parameter "DATA_FRAME" means data frame type; and the fifth parameter "2" restricts the data length of the current frame to two data bytes; and the last parameter "output_speed" stores the corresponding data bytes being transferred.

The return value retn represents whether the message is sent successfully.

The system is started when the corresponding M script is executed in the MATLAB environment. AutoCAN automation server and CANspider automation server are created when the M script initializes, the AutoCAN main interface shows up afterwards. The Simulink model is then loaded and the pre-defined simulation stages are carried out by the script. The control algorithm reads the signal, write variables and send frames, which realize the data exchange on the real CAN network among the virtual CAN nodes.

E. System Co-simulation Implementation

The CAN system is simulated in the AutoCAN environment, while the control algorithm is simulated in MATLAB/Simulink environment.

The CAN system simulation is symbol based, which uses a CAN communication database as mention above. The message transmission and reception of each network node is defined in the communication matrix of the

database. The transmission model of a CAN message object is usually divided into two kinds: cyclic and spontaneous.

The cyclic message is automatically sent by the corresponding virtual node and is scheduled by the simulation kernel on the CAN bus with a pre-defined interval, such as 10 milliseconds. Each message has its own transmission interval. If a node is active on the CAN bus, all its cyclic messages will be scheduled for the transmission. The data of the cyclic message can be continuously changed during each transmission, which is controlled by the corresponding virtual node. If some data need to be updated, the node has to modify the signal value in the message; it does not need to transmit the message immediately since this message is already scheduled and will be automatically sent in the upcoming communication cycle. To modify a specific signal's value, the function "set_signal" in the AutoCAN built-in AC language is used, take a signal called "PedalPos" as example:

set_signal(MSG_STATE, SGN_PEDAL_POS, val);

The first input parameter "MSG_STATE" is the message object which contains the signal "PedalPos"; the second input parameter "SGN_PEDAL" is the signal object to be modified, and the last parameter "val" is the destination value for the signal to be set.

Furthermore, the cyclic message is typically used to determine whether a specific node is active on the CAN bus during the simulation and test process.

Unlike cyclic transmission model, spontaneous message will not be automatically transmitted by the simulation kernel; the transmission of such kind of message is determined by the virtual node's internal algorithm. Events are typically triggered using such transmission model.

In summary, the CAN system simulation is based on the signal value refresh in cyclic messages and event trigger in spontaneous messages. When the CAN bus simulation is started in AutoCAN, all nodes will participate in the communication with their cyclic and spontaneous messages.

The control algorithm simulation is managed by MATLAB/Simulink, which is different from the CAN system simulation; however, there is a virtual CAN node associated with the control algorithm in the AutoCAN simulation environment, which provides a means to manipulate the cyclic and spontaneous messages for the control algorithm. During the execution of the control algorithm, when the algorithm needs to change a signal value in the cyclic message, the corresponding COM interface function in the AutoCAN automation server should be called.

The typical signal modification process is implemented as follows:

- Create a simulation variable in AutoCAN
- Invoke automation function "SetSimVarValue" to set the value of this variable from M-script in MATLAB
- Implement a "simulation-variable-change" call-back function in AC language in AutoCAN to handle the event when the value of this variable is changed
- In this call-back function, set the corresponding signal value with the AC function "set_signal"

There is no automation function like "SetSignalValue" in AutoCAN because the implementation of COM interface requires minimum coupling between different modules, there are only two functions available for data exchange: "GetSimVarValue" and "SetSimVarValue", which make the migration of application interface much easier.

The steps to create the co-simulation are as follows:

- Create a CAN database including all the virtual nodes' communication matrixes
- Implement a network topology in the AutoCAN simulation module
- Make AC script for each simulated node, so as to control the node's behavior during the simulation
- Implement call-backs to serve the requests from automation clients
- Implement automation requests in the M-script of MATLAB
- Run the simulation in AutoCAN and MATLAB

F. CAN Bus Disturbance during the Simulation

To guarantee the reliability and fault recovery ability of the control algorithm in the specified ECU, node test as well as network test should be performed. CANspider is used to disturb the message transmission on the network, and bring the specific ECU into bus off state.

A specific CAN disturbance can be performed with the help of MATLAB GUI in Fig. 3. The disturbance occurs when the "Start Digital Disturbance" button is pressed on the GUI.

However, to simulate the interferences in the vehicle environment, a timer is implemented in the M script to configure the CANspider hardware so as to bring the ABS node into off-line state at specific time intervals. Other network nodes are not influenced during the ABS bus off. The following functions published by the IDisturbance interface are used to configure the CANspider, which configures the disturbance settings and starts the disturbance:

```
retn = CANspider_Disturb_Specific_ID(ID_ABS, 32,
TIMEOUT_5S);
retn = CANspider_Start_Disturbance(ANALOG_DISABLED,
DIGITAL_ENABLED);
```

The function "CANspider_Disturb_Specific_ID" first configures the trigger source and disturbance type of the CANspider hardware, to make the CANspider hardware trigger only when the identifier of the ABS node appears on the bus. In this way the ABS node is disturbed while other CAN nodes continue to exchange the signals. The disturbance type is limited disturbance with counter equals 32, which allows only one bus off occur for a specific CAN node. There is also a timeout value in the

input parameters, which prevents the system from waiting endlessly when no frames occur on the bus.

After the successful configuration of the trigger source and disturbance type, the CANspider hardware is started by the "CANspider_Start_Disturbance" function, the input parameter "ANALOG_DISABLED" means no analog disturbance is applied, while "DIGITAL_ENABLED" indicates that the current disturbance type is digital disturbance.

Fig. 7 shows that the frame transmitted by the ABS is being disturbed by CANspider. The disturbance position is at the beginning of the Data Length field. A recessive bit is forced into dominant level, which results in the termination of the current frame transmission and error frame sent by the ABS node. After 32 times of successive disturbance, the ABS node enters bus off state.

When the disturbance is active, the messages being transmitted by the ABS are continuously disturbed and cannot be received by other network nodes. Error frames appear in the AutoCAN measurement windows, and the vehicle speed signal is lost. The control algorithm is estimated to tolerate this kind of error, enter error mode and try to recovery after the removal of the disturbance. With this test method, the stability and robustness of the ECU control algorithm can be fully tested.

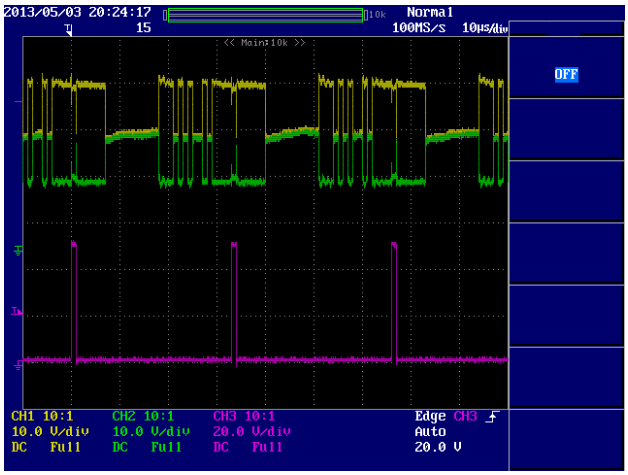

Figure 7. The message transmission of ABS is disturbed by CANspider during the simulation.

G. Automated Test Engine Implementation

In order to test all aspects of a control algorithm or multiple control algorithms in a reproducible manner, the test execution should be automated, a test engine is developed for the co-simulation-and-test system [11] [12].

The test engine is implemented as a plugin of AutoCAN in a DLL (dynamic link library) file, since AutoCAN has a plugin manager and is able to load external program as its own plugin, which makes automation more efficient.

An AutoCAN plugin is able to access the internal objects and functions of AutoCAN, such as message transmission and reception; and is also able to interact with other applications on Windows platform. To implement different test cases in order to test specific function points of an algorithm, the corresponding simulation environment should be different; this can be achieved by the implementation of test case selector, which selects the appropriate simulation configuration M-script file in MATLAB, and run the co-simulation based on the selected configuration.

To make MATLAB run a specific M-script file, the windows shell command can be used in the following manner [13]:

"MATLAB_binary_path\matlab.exe" -r configuration.m -logfile log_file_path

The file name of the shell command points to the path of the MATLAB executable "matlab.exe"; the "-r" means to run the M-script specified in the following parameter; the "configuration.m" specifies the M-script location on the disk, which can be changed in different test cases; and the "-logfile" parameter make MATLAB generate log file of the command execution in MATLAB environment; and the final parameter specifies the location of the destination log file. When the co-simulation-and-test is completed, the MATLAB application can be closed in the by invoking Windows API function - "CloseWindow".

The flowchart of the test engine is illustrated in Fig. 8.

The test engine first loads a list of test cases when it is started, and executes each test case one after another until all test cases are executed.

During the execution of automated testing, a specific test case is first selected by the test case selector, the runtime information is extracted from the test case, and MATLAB is invoked and the simulation is started based on the information from the test case.

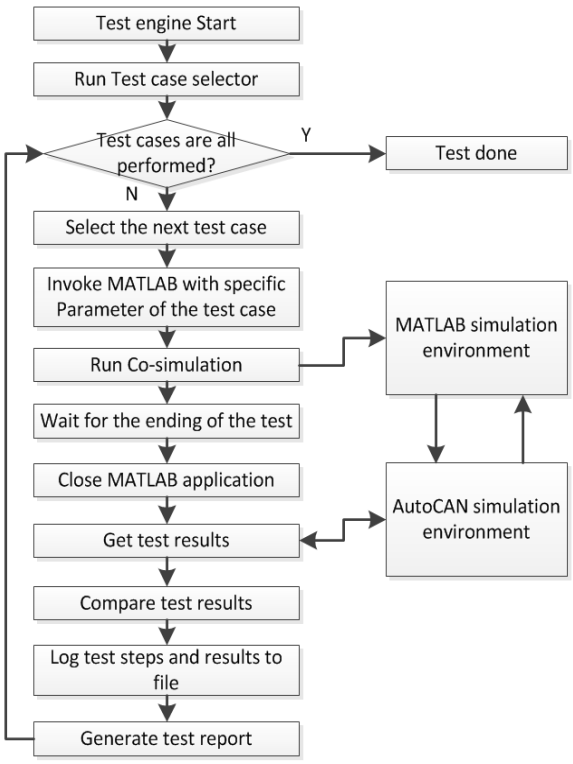

Figure 8. The flowchart of the test engine.

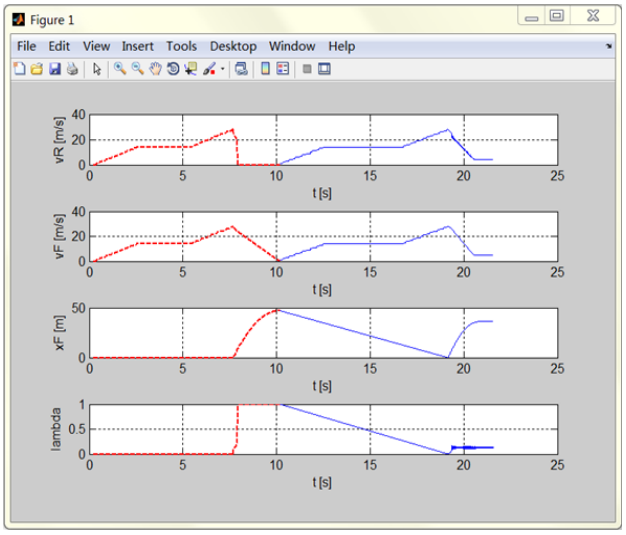

Figure 9. The test result curves.

The simulation in AutoCAN is then started by the M-script through the automation function "LoadConfiguration" and "StartSimulation".

The test engine is then paused and waits for the end of the co-simulation by monitoring a simulation variable in the AutoCAN environment, which is flagged when the MATLAB simulation ends. The MATLAB application is automatically closed by the test engine when the test case ends. The test results stored in the simulation variables are then retrieved by the test engine and compared with the expected value.

The test result is rated "FAIL" if there is one failure in the result check, otherwise, the test result is "PASS". The test data is logged into the log files and a test report is generated based on the log files.

IV. Result and Discussion

An ABS system control algorithm implemented in MATLAB/Simulink is tested in the Co-Simulation-and-Test system. The test steps are carried out in the M-script of MATLAB in the following order:

- Configure the CANspider digital disturbance module
- Run CAN bus simulation in AutoCAN
- Run control algorithm in Simulink
- Apply disturbances in the acceleration stage of the vehicle
- Compare the test results with the acceptance criteria

The test result is shown in four curves in Fig. 9.

In the result figure, vR is rear-wheel speed, vF is front-wheel speed, xF is braking distance and lambda is slip rate. Each curve is divided into four sections, which represent four stages in the simulation.

Stage 1 and stage 3 are acceleration processes, which are disturbed during the operation, meanwhile, the vF and vR fall into horizontal lines, for no messages can be received during the disturbance on the CAN bus. The speed and associated timestamp retrieved through the M script from AutoCAN trace window have the latest value before the disturbance and stay unchanged until the disturbance ends. The speed and timestamp signals are available again after the disturbance. The xF and lambda are not influenced in the stage 1 because they are not changed during the acceleration; they are either not influenced in stage 3 because they are simulation variables in the AutoCAN simulation model and are not influenced by the error in the real CAN bus. The disturbance only makes an impact on the CAN bus communication, the real operation in the actuators are not influenced, which means the vehicle continues to brake, thus the xF and lambda vary according to their original trend.

Test results show that the communication is also applicable between the CANspider and MATLAB so as to fulfill the test on the CAN bus system. The system under simulation is proved to be capable of recovering from bus failures, which meets the CAN bus performance characteristics.

V. Conclusions

The CAN bus co-simulation-and-test platform built based on AutoCAN, CANspider and MATLAB is able to simulate the input signals as the ones in actual automotive network system, and process them according to the control algorithm described by model in MATLAB/Simulink. The simulation data can be observed in real time. By means of COM technology, which enables one software to access the other one through interfaces, the resource of different software can be shared without code re-implementation. Compared with other solutions, the structure of the whole system is more flexible, the advantages of all the tools in the system, such as the modeling ability of MATLAB and the CAN bus simulation ability of AutoCAN and the CAN bus error inject capability of CANspider are combined, resulting in the programming efficiency enhanced and the development costs reduced.

To ease the simulation and test process, the MATLAB GUI is introduced, which interacts with the Simulink model and M scripts to accomplish the control over the data communication and CAN frame disturbances.

An automated test engine is implemented to execute a list of test cases for the specific algorithm, which automatically invoke MATLAB application based on different test configurations, runs co-simulation, and finally close MATLAB application and generate test reports. With the help of the test engine, the defect of the algorithm can be detected and reproduced.

The co-simulation-and-test is carried out on the basis of the actual automotive electronic system. Problems and defects of the control algorithm can be detected during the co-simulation-and-test period with the help of the CANspider. The experiments show the feasibility of the simulation and test method.

Moreover, the function of the system can be extended in this way, to achieve complicated algorithm simulation and test and hardware-in-the-loop simulation.

ACKNOWLEDGMENT

This work was supported by National High-tech R&D Program of China (863 Program) under grant No. 2011AA11A214.

REFERENCES

[1] Q. Alfio and F. Saleri, *Scientific Computing with MATLAB and Octave*, 2nd ed. Springer, 2006.
[2] J. Z. Ou and V. K. Prasanna, "MATLAB/Simulink based hardware/software co-simulation for designing using FPGA configured soft processors," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, 2005.
[3] J. G. Zhang, Y. L. Lei, H. B. Liu, X. P. Zhang, and Z. J. Liu, "Application of CANoe-Matlab co-simulation in DCT-CAN bus control," *Automobile Technology*, vol. 1, no. 9, pp. 7-10, 2010.
[4] H. Z. Guo, H. Chen, T. H. Song, and X. Zhou, "CAN nodes embedded automotive electronic throttle simulation system," *Journal of System Simulation*, vol. 21, no. 18, pp. 5716-5719, Sep. 2009.
[5] D. Box, *Essential COM*, Addison-Wesley, 1998.
[6] X. J. Li, G. Ye, Z. W. Li, and S. L. Ma, "Study and implementation of spacecraft integration test platform based on component technology," *Journal of Computers*, vol. 6, no. 5, pp. 963-968, May 2011.
[7] X. L. Bai and H. M. Zhang, "Design of digital filter based on VB and Matlab," in *Proc. 2009 9th International Conference on Electronic Measurement & Instruments*, Beijing, 2009, pp. 85-88.
[8] X. Y. Wang and R. D. Ji, "Design and implementation of simulation platform for fuzzy PID based on COM technology," in *Proc. 2011 International Conference on Control, Automation and Systems Engineering*, Singapore, 2011, pp. 1-3.
[9] J. Wang, J. Y. Chen, and Y. Zhuang, "Mixed programming between MATLAB and other programming languages," *Communications in Computer and Information Science* vol. 225, pp. 669-676, 2011.
[10] D. Chappell, *Understanding ActiveX and OLE*, Microsoft Press, 1996.
[11] S. Wang, Y. D. Ji, and S. Y. Yang, "A micro-kernel test engine for automatic test system," *Journal of Computers*, vol. 6, no. 1, pp. 3-10, January 2011.
[12] J. Yang, B. Liang, T. Zhang, J. Y. Song and L. L. Song, "Laboratory test system design for star sensor performance evaluation," *Journal of Computers*, vol. 7, no. 4, pp. 1056-1063, April 2012.
[13] C. G. Wang, J. F. He, G. X. Li, and J. W. Han, "An automated test system for flight simulator fidelity evaluation," *Journal of Computers*, vol. 4, no. 11, pp. 1083-1090, November 2009.

Chu Liu born in Fujian, China in July 1985. His area of research is automotive electronic network. He received his Master's degree in Bachelor of Engineering from the Tongji University in 2011 in Shanghai, China. He is a research student in Tongji University in Shanghai, China. His research interests include: Automotive Network and Intelligent Vehicle.

