Simulation Framework Supporting Model Based Development of Automotive Software Applications

[1]Shruthe R, [2]Paul Perumattathu, [3]Naveen Kalappa, [4]Sankar Dasiga
[1]MTech. Student, [2]Sr. Software Engineer KPIT, [3]Principal Architect KPIT, [4]Professor
[1]Department of Electronics and Communication, Nitte Meenakshi Institute of Technology, Bangalore, India

***Abstract*: Software development practices that are cost effective and swift are need of the hour in automotive industries. Simulation is a very effective technique that helps in recreating an actual scenario and enables analyzing and visualizing of a system's behavior in that scenario. This in turn helps in reducing the number of changes that are required to be done at the time of hardware testing as the short comings of the control algorithm are known well in advance which reduces the development cost and results in faster time to market. In this paper we describe the methodology used to develop a simulation framework that aids model based development of automotive body systems. Here, we have applied this methodology to develop the framework for Power window and door lock.**

***Index Terms* - Controller Area Network, Functional Mockup Units, Model Based Development, V-Model, Modularity, SWIG, Database, Visualization.**

I. INTRODUCTION

A robust simulation framework plays a key role in enabling detailed analysis of a system's behavior. A major requirement to be able to test a system in a simulated environment is a plant model that comprises all mechanical and electrical components of a system. Next, a virtual model of the system is needed that allows us to visualize the functionality of system. A GUI to allow interaction between the simulator and user is also necessary.

Sameer M. Prabhu and Pieter J. Mosterman have suggested the need for systematic approach in product development "Given competitive temporal and cost constraints, developing a product on time and within budget requires a systematic approach to design and realization" [1, p.1]. A good simulator is an important component of any testing.

Simulation has numerous advantages – It allows assessment of a scenario in an efficient manner as the systems response will be known well in advance, a virtual environment is void of risks and is safe for testing, animation/visualization makes it easy to understand concepts, enables virtual integration of models developed by different developers without having to worry about compatibility, simulation can be stopped mid-way and can be re-simulated with different parameter values which is difficult with an actual test rig.

This paper describes the methodology used in development of a simulation framework for testing of Power window that was built using open sources tools – Python, Open Modelica and FreeCAD. The features present are as follows:

1. Testing of control algorithm along with the plant model modelled using OpenModelica and Simulink.
2. Visualization of plant model using CAD models developed using PythonOCC.
3. Enabling control of plant model through vehicle network, here, Controller Area Network (CAN).
4. Storage of simulation data using the feature of database storage.
5. Import and testing of Functional Mockup Units (FMU) using FMI feature.
6. Testing plant model with varied parameter values.
7. Framework and System level modularity to enable reuse of code.

II. MODEL BASED DEVELOPMENT

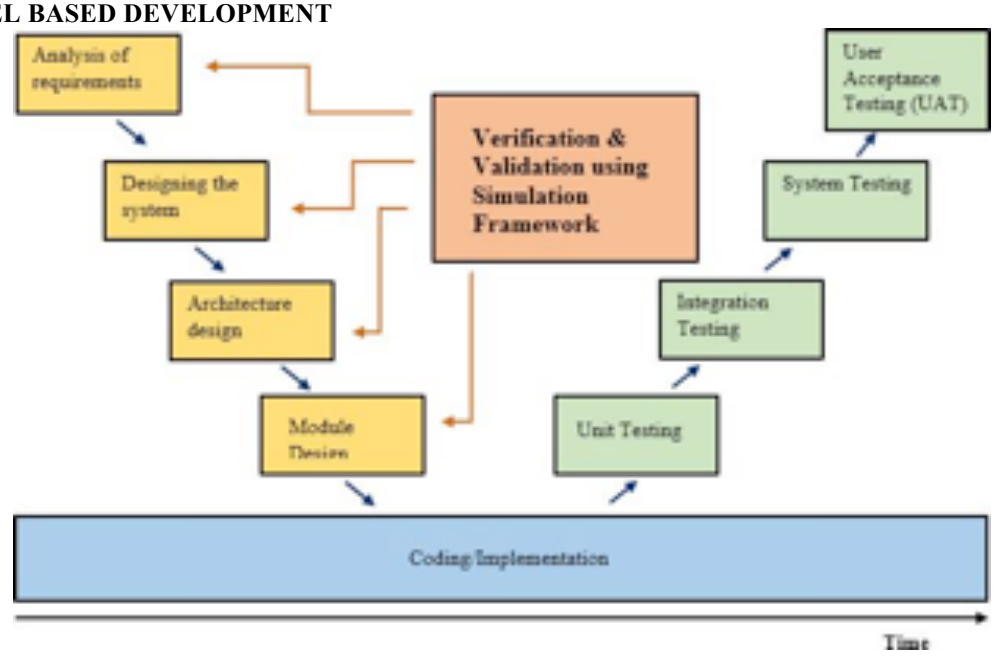

Fig. 1 V-model with simulation included

An Embedded Software designed on the basis of Model-Based Design (MBD) helps in validating and verifying the system that is being developed at every stage of the development process.

MBD uses the concept of visualization. This helps in easily communicating throughout the design cycle with the help of plant models developed with the help of simulation tools. A Plant model is made up of blocks interconnected to form a physical model that is similar in behavior to the actual system. It is administered using a mathematical algorithm/model called as controller which is developed using raw data available from the real-world. The response of the model along with controller with respect to time is tested virtually using simulation. Once the functionality of plant model with the controller is verified and validated against the given requirements the controller is ready to be deployed on to hardware.

The concept of model based design revolves around "V-model". V-model describes the steps involved in software development which are represented across two sides of a V, one side represents the verification phase to check if requirements are met and the other side represents the validation phase which demonstrates that the requirements have been met. Flow used in V-model is as shown in the Figure 1.

With the inclusion of simulation in MBD verification and validation processes are completed well in advance. Due to this, short comings in the control algorithm can be recognized and modified so as to match the requirements that were put forth during requirement analysis stage. Because of this, the outcome of verification and validation done via simulation results in a near to perfect control algorithm that can be deployed on to hardware with very few alterations. This reduces the time taken for software development which in turn minimizes the time to market. Also, simulation provides a wider scope to improve the quality of control algorithm as the number of times it can be tested are countless as opposed to that of physical testing where the number of re-runs are limited.

The Simulation framework described in this paper is aimed at speeding up the time taken to develop software based on the idea illustrated in Figure 1.

III. SIMULATION FRAMEWORK

Basic Concept

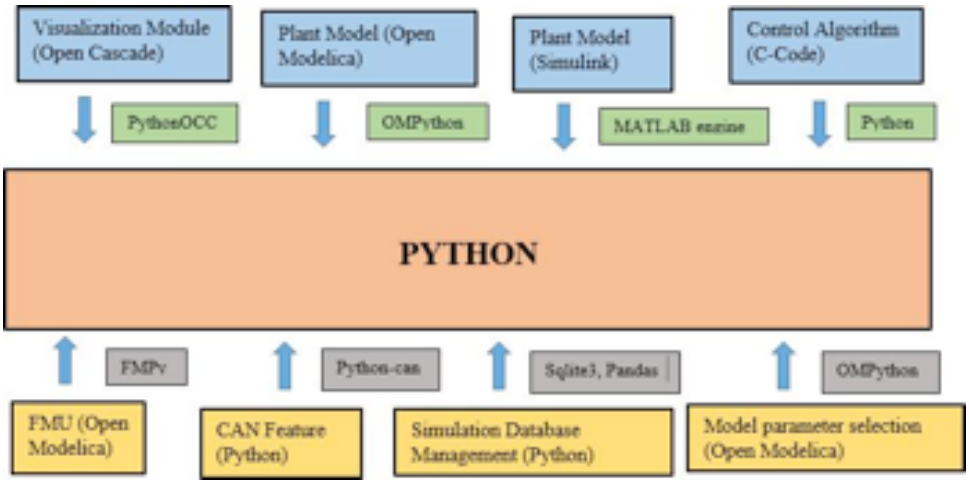

Fig. 2 Block diagram depicting concept of the simulation framework developed

The simulation framework described in this paper contains a GUI, plant model and a CAD model that provides required visualization. The foundation of this framework is Python. Python was chosen due to the diversity of packages for varied functionalities supported by it.

The efficiency and robustness of simulation software should be continually enhanced to bring it on par with the tools already present in the market. The features that were added to the framework were chosen upon after assessing the attributes of various automotive simulation tools present in the market. Control Desk and Automotive Simulation Models (ASM) by dSpace, Canalyser and Canape by Vector, Silver by Qtronic are a few tools that support the features listed earlier.

This framework acts as a platform for model based testing and validation of any system whose plant model, control algorithm and 3-D CAD models can be developed. The work presented in this paper focuses on automotive body systems, mainly, Power window and door lock.

The framework houses numerous components which were included into the framework using Application Programming Interfaces (APIs). APIs pave a path of communication between two different software environments. Here, Python packages acts as an API for interaction between the different applications that are used for development. The components that present, shown in Figure 2, are as described below –

Plant Model – A plant model composed of electrical and mechanical components were assembled into an electrical circuit using Openmodelica Connection Editor by making use of the available libraries. This plant model could be simulated using OMPython which is a package available in Python. The values of variables can be fetched using OMPython and these values can be plotted using Matplotlib.

User Interface – Python's Tkinter was used to develop an interactive GUI consisting of buttons which can be programmed.

Through these the user can apply inputs and connect with the plant model.

Visualization Module – FreeCAD was used to create a CAD model of power window and door lock. It was then linked to the framework using PythonOCC. This was used to better visualize the plant model.

Control Algorithm – The framework can be used to verify if the behavior of control algorithm given equals the expected behavior of the system. A SWIG (Simplified Wrapper and Interface Generator) wrapper code was written to enable communication between the control algorithm (a C code) and python framework.

Simulink – Plant models consisting of electrical and mechanical components developed using Simulink can also be simulated using the package MATLAB Engine.

Functional Mock-Up Units (FMUs) –A FMU of the plant model was generated using Openmodelica and loaded on to the framework using Python's FMPy package which allows simulation of FMUs. The plots were obtained using python's matplotlib and was used to compare with that of ordinary plant model and was found to be the same.

Controller Area Network (CAN) - Python-can package was used to control the plant model by sending CAN messages each time the user pressed a button.

Simulation Database – The CAN messages that are sent are logged on to a database and can be retrieved for analysis if needed in case of any fault.

Model Parameter Selection – The parameters of plant model can be varied via GUI and the changes that happen can be realized with the help of simulation results obtained.

Tools used

The framework was developed using open source tools available in the market. The tools used are as follows:

PyCharm Community Edition – Integrated Development Environment (IDE) for Python.

OpenModelica – For developing plant models of the system.

FreeCAD – For visualization using 3-D CAD models.

IV. FRAMEWORK FEATURES

The diagram in Figure 3 depicts the flow of the framework.

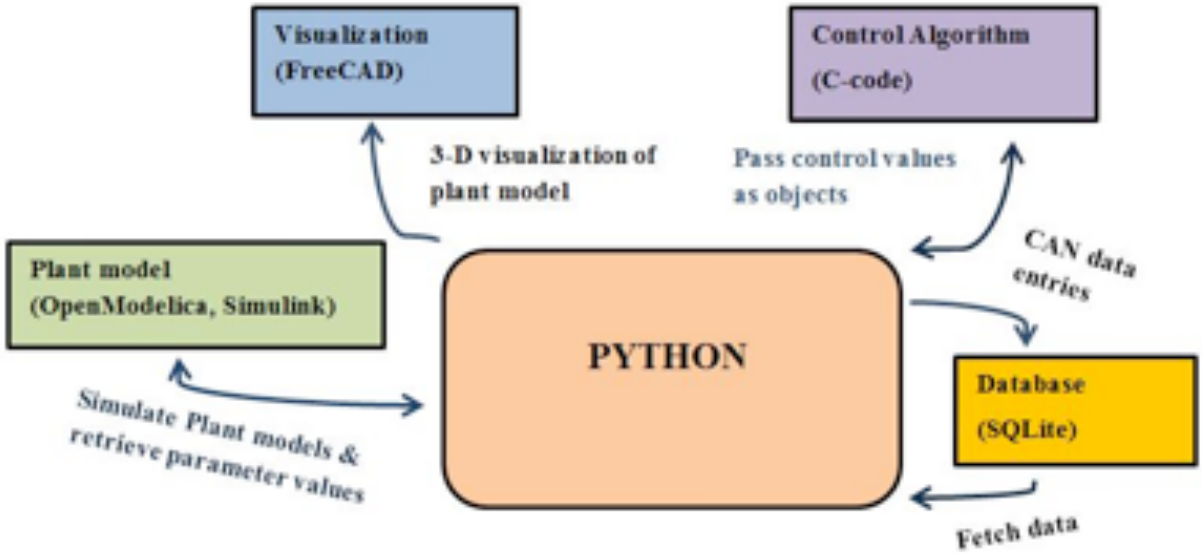

Fig. 3 Framework Flow

Feature 1 – Plant Model Simulation

In MBD, a plant model consisting of electrical and mechanical components which emulates the behavior of the system that is modelled is an important component. Plant Models help in the verification of a system even in the absence of hardware. Although it cannot completely imitate the functionality of an actual system the fidelity of the plant model can be improved in order to almost match the real system.

In our case, a basic plant model of Power window and door lock was modelled using OpenModelica. OpenModelica comprises of a number of readily available libraries with electrical, mechanical, hydraulic, electro-mechanical etc. components. These were interconnected to develop the required plant model as shown in Figure 9.

This plant model was loaded on to the python framework using OMPython which contains a method named ModelicaSystem. This method allows importing an OpenModelica plant model and retrieving the values of parameters required. The values obtained are then processed using Matplotlib to visualize the results in the form of plots.

Using the concept described above, Plant models were also developed using MATLAB's Simulink as shown in the Figure 4. The Simulink plant model was loaded onto the framework using python's MATLAB Engine. MATLAB Engine's matlab.engine.start_matlab allows loading the model into python environment and testing the model. The plots obtained through matplotlib can then be used to visualize the results.

Feature 2 – Visualization Module

Visualization is an important component of Simulation. Simulation and visualization fused together help in developing a high quality product with a high grade of precision in terms of functionality. Additionally, visualization enables the simulation software's user to easily grasp the concept. Thus, in our framework we have incorporated visualization in terms of plots, CAD model and animation.

First, a 3-D CAD model of Power window and door lock was created using FreeCAD. FreeCAD is an open source 3D modeler. It is used for mechanical engineering based product design. FreeCAD and its functionality can be extended to python by using PythonOCC. New objects can be added and programmed exclusively using Python. It has plugins to Python and C++ and a built-in python interpreter.

The CAD models were then converted to STEP (Standard for the Exchange of Product model data) files. Converting a CAD model to STEP file makes it easy to exchange these models between various CAD system users. The CAD models made are as shown in Figure 6.

These files were added to python using PythonOCC's StepImporter. The loaded models were then animated using PythonOCC to duplicate the real-world visual aspects of window and door lock. Animation makes it easy-to-understand even for people with least technical comprehension. The animations were then mapped to the event of button press depending on the functionality required to the user which is known via GUI.

Feature 3 – Control Algorithm

This framework was aimed at enabling verification of control algorithm essentially coded in C. A control algorithm must be able to predict the system's outcome based of the inputs given to the plant model sensors.

The interaction between Python and C-code happens through Simplified Wrapper and Interface Generator (SWIG). SWIG connects a code written in C or C++ to other high level languages like Pearl, Python etc. It provides an interface to test software developed using C or C++. The source code generated via the tools helps to pass data values between the two different programming languages.

In our case, a control algorithm was designed to be able to understand the intention of button press i.e. to move the window up or down. When a button press on the GUI (event) is detected python passes a value containing the details of button press to the control algorithm via the glue code. Depending on the value of the object received the control algorithm decides which window the user has intended to move and the direction of movement expected. Once, the inputs given are processed the object is sent back to the python framework. Based on the values received back python invokes the corresponding functionality which can then be seen visually.

Feature 4 - Control of plant model using CAN (Controller Area Network)

CAN is a protocol used for serial communication. It was developed with an intention of reducing the number of wire harnesses present in a vehicle. It is a multi-master protocol that enables different ECUs to communicate with each other. Each ECU is considered as a node. CAN enables exchange of signals between sensors and ECUs. The messages that are being transmitted have a priority. The lowest number has the highest priority. All the messages transmitted across the bus are broadcasted. Depending on the arbitration ID the message is accepted by the node the message was intended for.

Being able to control the plant model using CAN helps in bringing virtual model closer to the actual system in terms of functionality. Here, we've used python-can package to achieve this. Python-can provides a number of interfaces like pcan, socket can, kvaser, virtual can etc. Here, virtual CAN has been used to test this feature. Virtual CAN helps in transmission and reception of CAN messages even when a CAN hardware is not available. When a button in GUI (Up/Down/Stop) is pressed it is programmed to send a CAN message which is received by another bus. An assert statement checks for equality of the sent and received message following which the required functionality is performed via a function call.

Feature 5 - Storing simulation data using database feature

As stated by CIMdata that "Indeed, the threat is that simulation data will simply overwhelm us, and become a significant source of waste" [3, p3] which requires that useful data needs to be extracted out of all the data that gets generated during every simulation cycle. Although all the data that is generated may not be useful the ones that can prove to be useful for further improvements in the plant model can be stored in a database file or an excel file. In our case, each time an event i.e. a button press takes place and the corresponding CAN message is sent, the message that is sent along with corresponding time and date is stored in an excel file. This data can then be used in case a malfunction is noticed in the functioning of window for analysis.

Using Python's sqlite3 package the data pertaining to button press was added to a database file. The contents of this database file were then moved to an excel sheet using python Pandas for clear presentation of data stored in database. A button present in the GUI was programmed to print the contents of data on output console. For this, the data in excel file was converted into data frames using pandas excel read and the data frame thus printed gets printed on the console. This helps us to access database/excel contents without having to look for the required excel file.

Feature 6- Import and testing of functional mockup units (FMUs)

Functional Mockup Interface (FMI) is a standard widely used in computer simulations. It's based on the theory that just like a real system is constructed by combining different components that interact with one another using laws of physics a virtual model of the same system can also be made by combining different components that also can interact with each other. An FMU that is generated in a particular environment can be simulated and tested in another environment and thus FMI helps in creating a co- simulation environment. It provides a link/interface between two different simulation programs. Also, the same model can be reused for different purposes using different tools.

There are two variations of FMUs that are available – Model Exchange (ME) and Co-Simulation (CS). In ME, tool that imports the FMU provides a solver which decides on time steps at which output needs to be computed. FMU itself will contain inputs. In case of CS, the solver is implanted into FMU by the exporting tool. The tool that imports the FMU can only set inputs and ask FMU to move to next time step and calculate corresponding outputs. Since the components present in a FMU model cannot be seen it helps in maintaining commercial and industrial confidentiality especially in projects that involve joint development from different companies. This necessitated inclusion of FMU feature to this simulation framework.

In our case, a FMU of an Openmodelica plant model was generated (FMI version-2.0 and FMI type-Model Exchange and Co-Simulation). It was then simulated using python's FMPy package which allows simulation of FMUs by acting as an interface between python and the generated FMU. The results obtained were verified and compared with the results obtained after simulating regular plant model and both were found to be same.

Feature 7 - Testing plant model with varied parameter values

In any simulation the results obtained will be verifiable for only that particular value of a parameter. To be able to convert these results into analytic results the simulation framework must empower the user to be able to simulate for different values of parameter. This allows the user to test and validate the system's behavior for varied parameter values.

In the presented work, values of supply voltage, nominal speed and nominal current can be changed and the resulting value of current can be visualized via matplotlib plots. The values of parameters mentioned previously are changed using support of OMPython which has a method that sets value of parameters. The values to which parameters are to be positioned are based on input given using GUI after which the model is simulated and plot is obtained. The plots can be used to evaluate the changes that occur in the model output (here, motor current) for changes that are made.

Feature 8 - System and framework level modularity to enable reuse of code

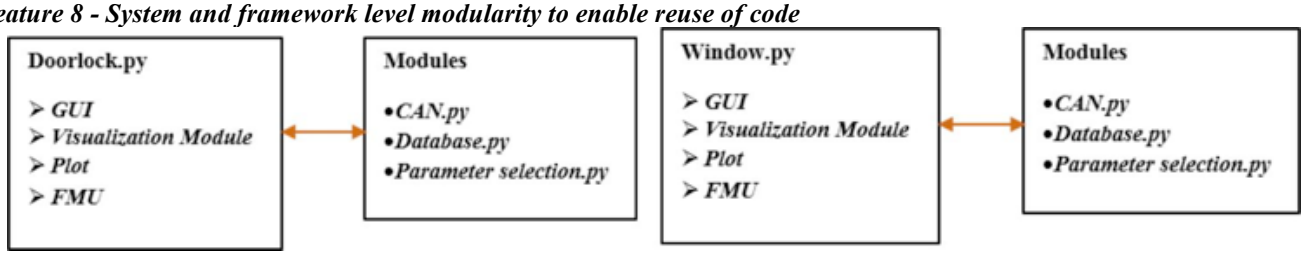

Fig. 4 System Level Modularization

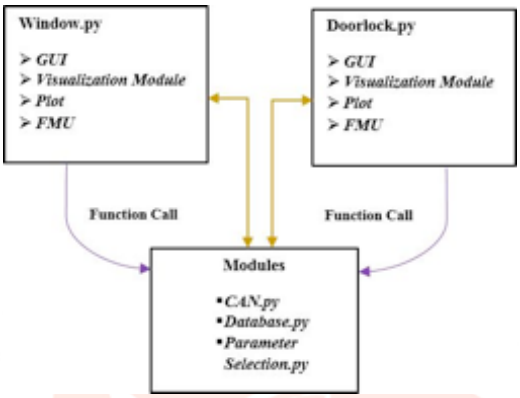

Fig. 5 Framework Level Modularization

Modular programming is a software design technique wherein a huge chunk of code written for a particular application is fractionated into several fragments. These fragments' are called as modules and each of them perform distinct functionalities. They can have no or very less reliance on other modules. Interactions between modules happen through function calls and parameter passing. The final application that can be executed is obtained by connecting all these modules. Modularization allows rapid software development as each module can be developed independently even with slightest knowledge about rest of the application. Modules of an application can be utilized in some other application if required. A piece of code is executed successively. Modularization brings in Parallelism entire code remains active at all times. Programmer can decide which part of the code he wants to include in 'always active' part. In python modules can also be run as a separate script provided it has no functional dependency with other modules.

Here, Modularity was accomplished by first identifying segments of code that could be modularized. In our case CAN, Database and Model parameter selection could be segregated as separate modules. The main code consists of GUI, CAD and plot parts. The modules separated interact with the main code through function calls and parameter passing. This part can be labelled as "System Level Modularity" as shown in Figure 4.

Modules are imported to the main code. Each time a button is pressed data regarding button press is passed as a parameter during function call to file containing CAN module, also, a function call to database module's file for adding data to corresponding database table is made. In the case of parameter selection, a button in GUI was created which the user can click when they want to simulate the plant model with varied parameter values. If a button press is recognized, it invokes a function call within the main function that runs the parameter selection module as a separate script.

Once the framework for door lock was made all the features that were collectively shared by the two frameworks i.e. CAN and Database was made into two modules and was collectively shared by both the frameworks as depicted in Figure 5. This is called as "Framework level modularity".

VI. OUTPUTS

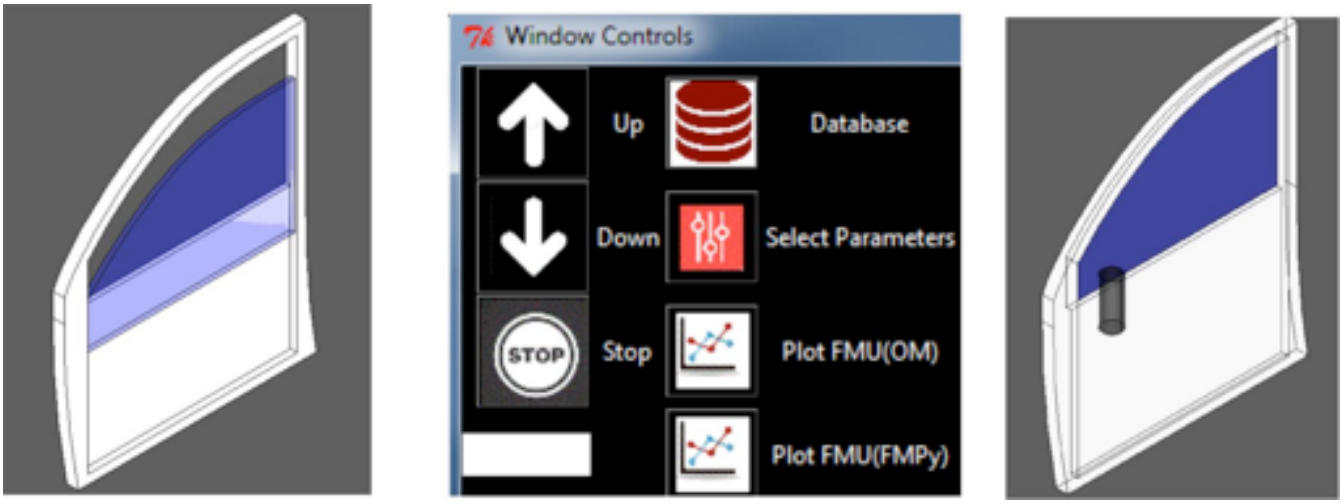

Fig. 6 GUI and CAD models for controlling and visualizing of door lock and power window

```
Message sent on Virtual bus channel test
Message recvd on Virtual bus channel test
Timestamp: 1556019652.647000    ID: 00000000    X        DLC: 1    75

                                                                    Message sent on Virtual bus channel test
                                                                    Message recvd on Virtual bus channel test
                                                          'u'       Timestamp: 1556019591.842000    ID: 00000000    X        DLC: 1    6c                    '1'
```

Fig. 7 CAN messages that get sent on a button click

	can_msg	recv_time		can_msg	recv_time
0	Unlock	2019-04-19 11:45:51	0	down	2019-04-19 11:42:24
1	Lock	2019-04-19 11:45:54	1	stopped	2019-04-19 11:42:26
2	Unlock	2019-04-19 11:45:54	2	up	2019-04-19 11:42:26
3	Lock	2019-04-19 11:45:55	3	stopped	2019-04-19 11:42:27
4	Unlock	2019-04-19 11:51:17	4	down	2019-04-19 11:44:01
5	Lock	2019-04-19 11:51:17	5	stopped	2019-04-19 11:44:02
6	Unlock	2019-04-19 11:51:47	6	down	2019-04-19 11:44:58
7	Unlock	2019-04-22 07:51:57	7	stopped	2019-04-19 11:45:00
8	Lock	2019-04-22 07:51:58	8	up	2019-04-19 11:45:01
9	Unlock	2019-04-22 07:51:59	9	stopped	2019-04-19 11:45:01
10	Unlock	2019-04-22 07:54:03	10	down	2019-04-19 11:45:22
11	Lock	2019-04-22 07:54:03	11	stopped	2019-04-19 11:45:26
12	Unlock	2019-04-22 07:54:04	12	down	2019-04-19 11:47:43
13	Unlock	2019-04-22 07:55:39	13	stopped	2019-04-19 11:47:44
14	Lock	2019-04-22 07:55:40	14	up	2019-04-19 11:47:45
15	Unlock	2019-04-22 07:55:41	15	stopped	2019-04-19 11:47:45
16	Unlock	2019-04-22 07:56:21	16	down	2019-04-22 07:58:02
17	Lock	2019-04-22 07:56:21	17	stopped	2019-04-22 07:58:04
18	Unlock	2019-04-22 07:56:22	18	up	2019-04-22 07:58:05
19	Unlock	2019-04-22 08:05:07	19	stopped	2019-04-22 07:58:05
20	Lock	2019-04-22 08:05:08	20	down	2019-04-22 08:31:14
21	Unlock	2019-04-22 08:06:26	21	stopped	2019-04-22 08:31:16
22	Lock	2019-04-22 08:06:27	22	up	2019-04-22 08:31:16
23	Unlock	2019-04-22 08:06:28	23	stopped	2019-04-22 08:31:18
24	Unlock	2019-04-22 08:08:31	24	down	2019-04-22 08:58:46
25	Lock	2019-04-22 08:09:17	25	stopped	2019-04-22 08:58:47
26	Unlock	2019-04-22 08:09:18	26	up	2019-04-22 08:58:49
27	Lock	2019-04-22 08:09:19	27	stopped	2019-04-22 08:58:49
28	Unlock	2019-04-22 08:10:05	28	down	2019-04-22 09:28:53
29	Lock	2019-04-22 08:10:06	29	stopped	2019-04-22 09:28:55
..	...	...	30	down	2019-04-23 08:58:58
46	Unlock	2019-04-22 08:33:42	31	stopped	2019-04-23 08:58:59
47	Lock	2019-04-22 08:33:43	32	up	2019-04-23 08:59:00
			33	stopped	2019-04-23 08:59:01

Fig. 8 Data of the button pressed, along with date and time is updated in an excel sheet

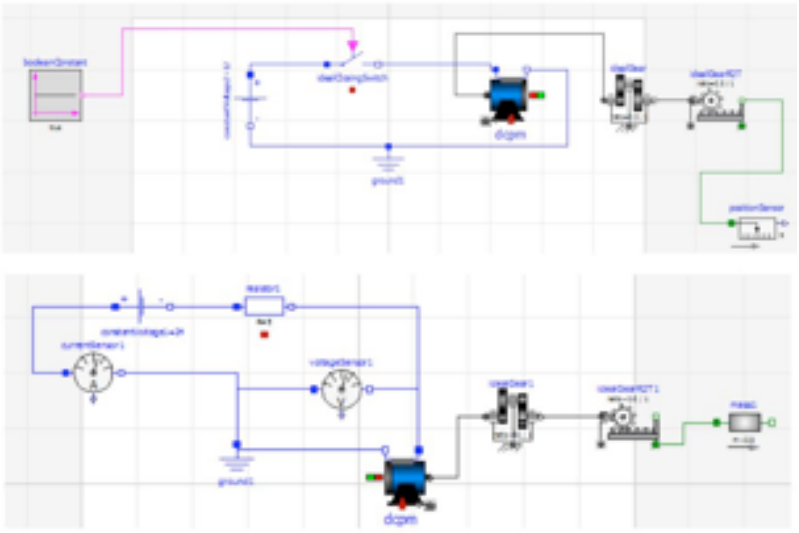

Fig. 9 Openmodelica model of Power window and Door lock

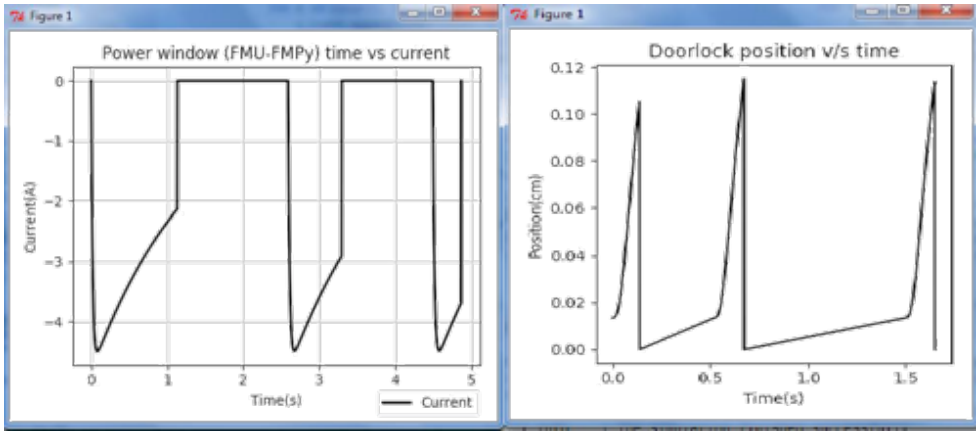

Fig. 10 Plots Obtained

VII. CONCLUSION

This paper talks about the features that have been added to the simulation framework of a Power window. It elaborates how these features prove to be useful. The framework has also been extended to car door lock along the same lines. Now, the framework incorporates the CAN feature and also the simulation data will be saved in a excel sheet and can be retrieved for further analysis. Also, the user can vary the model parameters such as motor speed, current etc. and check the corresponding changes through a plot. Functional Mockup Units (FMUs) can also be tested. System level and framework level modularization has also been incorporated for both the frameworks. The overall framework with all the incorporated features is as depicted in Figure 2.

VIII. REFERENCES

[1] Sameer M. Prabhu and Pieter J. Mosterman, "Model-Based Design of a Power Window System: Modeling, Simulation and Validation" in a Mathworks lecture on HS3

[2] Robert Bosch GmbH, "CAN Specification", Technical Report, Robert Bosch GmbH, Postfach 3002 40, D-70442, Stuttgart, Germany, 1991.

[3] CIMdata (2012), "Beyond Simulation Data Management, Why Innovative Approaches are required", Document, CIMdata, 3909 Research Park Drive, Ann Arbor, MI 48108.