AutoPlug - Open Architecture for Plug-n-Play Automotive Services

Dept. of CIS - Senior Design 2010-2011

Ross Boczar
boczar@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Gabriel Torres
gtorres@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Jason Suapengco
jasonns@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Rahul Mangharam *
rahulm@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

In 2009, 15.2 million vehicles were recalled in the United States, and of those, 1.3 million were recalled by General Motors due to software issues alone. This recall resulted in more than 136 million dollars in losses for the firm. There is a need to remotely diagnose, update and certify automotive software for efficient recall and safety management. This proposal will discuss the ability to build an automotive Electronic Controller Unit (ECU) test-bed to develop mechanisms and protocols for remote diagnosis, programming and testing of future vehicles. Using this electronic test-bed, a case study will be conducted demonstrating the viability of remote diagnostics and software management via AutoPlug. The ECU test-bed can also be used as a standard for other testing to occur on the modern car’s electronic system and opens doors to third-party development.

1. INTRODUCTION

In 2009, vehicle recalls in the United States crossed over 15 million, with many due to software bugs [15]. Consequently, there is a need to provide an efficient means to remotely diagnose, update and certify automotive software for efficient recall and safety management. As computers are becoming more integrated into cars, the amount of code put into that car increases tremendously. The modern car is not the purely mechanical device it once was - now having somewhere between 70-100 Electronic Control Units (ECUs) networked throughout the car. These 70-100 ECUs can contain upwards of 100 million lines of code, a number which is increasing since cars will only get more complex [8]. As a point of reference, Microsoft’s Windows XP contained about 40 million lines of code, and it was released in 2001 [16].

Due to this increased reliance on electronics, automobile repairs are no longer limited to mechanical problems, with repairs due to electronic or software bugs becoming more widespread. Since it is hard enough to detect one error in 10,000 lines of code [10], the amount of errors in 100 million lines of code must be significant. Although these errors may not always be fatal, many of them can lead to safety risks or recalls which is why automotive software management is crucial. As the amount of errors in a car become apparent, car companies are required to recall millions of cars resulting in millions of dollars in losses. By creating an efficient way to manage warranty remotely, these recalls can be avoided and those millions saved.

AutoPlug aims to assist manufacturers in this area, by designing an efficient way to correct software problems after release. AutoPlug is not a system that aims to perform diagnostics better, but rather allow manufactures to discover errors with ease and more importantly, reduce the costs of resolving said errors. AutoPlug consists of three layers: simulation, ECU and middleware. The simulation layer will model a real automobile, which will provide a form of verification for the ECU network. The models used for simulation will come from Torcs, an open source racing simulator [5]. This simulation will connect to the ECU network through data acquisition boards. The ECU network will consist of several micro-controllers, each of which will perform a specific function (e.g. steering, door locks, accelerator, etc.). These micro-controllers represent the current ECUs that are in place in cars today. Some functions of vehicles (such as the anti-lock braking system (ABS) or differential) require several ECUs to work together in order to perform the necessary tasks, thus there is a need for them to communicate. To tackle this problem the micro-controllers will be networked together using the industry standard Controller Area Network (CAN) protocol. The CAN protocol is used in virtually all modern automobiles as a means of communication between ECUs. Finally, the “middleware” layer consists of a laptop computer that will not only perform the more complex tasks that individual micro-controllers would be unable to do, but also act as an interface between the ECU network and the car manufacturers.

2. RELATED WORK

The motivation behind creating an automotive ECU test bed with mechanisms for remote diagnosis and software management is based on the idea of the “car as a platform” referenced in [14] as well as other sources. We will first examine projects whose goals are similar to that of this proposal’s. A paper on AutoSar [11], a project dedicated to creating
a standard for an automotive architecture, concludes how the introduction of an industry wide standard is necessary and will open new and different business opportunities for OEMs and their suppliers. As a collaboration of many different manufacturers (BMW, Bosch, Continental, Crysler, Ford, GM, Toyota, and Volkswagen), the goals of this paper can be seen as profitable for these companies. Although AutoSar attempts to standardize hardware components in vehicles, AutoPlug is unique in that it concentrates on only the electrical network. Nonetheless, AutoSar did reveal how such standardization may be profitable for automobile manufacturers.

Vehicle Simulation Layer. Vehicle dynamics simulation could be considered its own industry within automobile design. Many commercial off-the-shelf packages exist for simulating real-time motion of automobiles. CarSim [3], rFactor [4], and ADAMS/Car [1] are some of the vehicle simulation suites that have the ability to integrate with controller models for real-time simulation. There is also literature on different approaches to vehicle dynamics simulation, including multi-body [6] and analytical models [12].

For the purposes of AutoPlug, it is not necessary to design a car model simulator from scratch, as that would be a poor allocation of resources and would not contribute much to the overall success of AutoPlug. Instead, existing models built into Torcs will be used and adapted to fit the needs of the middleware and ECU layers. By adapting these preexisting models, a lightweight simulator that will prove the feasibility of the ECU network will be created.

With respect to models of individual vehicle components or ECU controllers (such as the engine, ABS system, or steering control), a similar approach will be taken. There exist many examples of these systems, such as [13], and these will be pieced together and possibly modified to fit into the overall automobile model. However, with respect to ECU controller software, a more design-based approach may be taken if the dynamics simulation portion of the model is completed relatively quickly.

Middleware Layer. There are several systems, such as General Motors’ OnStar, that have features that service the car remotely [14]. These systems do not perform firmware upgrades nor allow for direct remote management of the firmware. Additionally, these programs are company specific and do not adhere to a standard across manufacturers. The diagnostic system and upgrade being proposed would be available for cars not outfitted with an OnStar type system. Although OBD-II ports are not installed in all cars, they were able to obtain read/write access to an ECU test bed, but used an actual vehicle in their security research goes on to reveal how it may be possible to use fewer processors to run jobs while matching the required time constraints by utilizing the processors in an effective manner. It is important to keep in mind this idea of a time constraint in the ECU test bed in order to ensure safe functioning of the car.

In addition to research about job scheduling across CAN, there can be job scheduling in an ECU [18]. This research uses ECUs ported to an OSEK real time operating system (RTOS)- similar to this proposal’s. Using OSEK, job speeds are reported and the time it takes to switch tasks are recorded. The results show that the OSEK RTOS was able to complete the many tasks required of an ECU of an automobile in an efficient manner.

3. SYSTEM MODEL

The purpose of this project is to create, in a sense, a ‘Programmable Vehicle’ by which mechanisms will be created for remote diagnostics and software management. This
'Programmable Vehicle' takes the form of an Electronic Controller Unit test-bed networked using the CAN protocol. The test-bed will be connected to models for the simulation of a physical car and a laptop gateway for software management. The middleware gateway allows for efficient recall management, the main motivation in this project. In addition to recall management, the ECU test-bed can be used as a standard for other automotive testing.

At the most abstract level, AutoPlug can be viewed as the interconnection of three different layers: the simulation layer, the ECU-CAN network layer, and the middleware layer. User inputs are passed directly to the ECU testbed as they would in a real car. The individual controllers in the ECU testbed then process this data which is then passed to the car simulation model. Once the car simulator models the physical car it then outputs simulated sensor data about the car to the network of microcontrollers representing the ECU network. The middleware layer can then read the ECUs CAN Network and process the data to display car information, perform diagnostics and make decisions about the car’s operation. Figure 1 illustrates an abstract view of the interaction that occurs between each of the three layers.

3.1 Simulation Subsystem

The core of the simulation subsystem is a simulation of an automobile using car models from Torcs. As an open source racing simulator, Torcs contains models in C that will simulate real time input for each component of the ECU testbed. Since our project by nature intends to create an open-source architecture, we need to use an “open” car simulation model, as having to reverse-engineer an actual car in order to interface it with the other components of our architecture is the exact problem we are trying to avoid.

The automobile model will simulate car dynamics, as well as individual car subsystem components (engine, suspension, steering system) and models for the sensors and actuators transmitting and receiving data to and from the ECU network. In general, the model will attempt to integrate and build on as many “verified models” as possible. This will allow the design focus to be placed on the ECU network-middleware interaction, which is the central focus of AutoPlug.

This car model will not be based off of one specific car, but will have the general class of “luxury sedan” (e.g. BMW 3-Series, Acura TL). The reason for the model taking this form is that the AutoPlug open architecture is a proposed system for implementation on future cars. Also, luxury sedans typically have a more electronics and software components than average.

Using available resources, the implementation of a chosen car model will attempt to reflect the specifications of a typical car in that class. However, there were undoubtedly cases where specific data on some car component were not available. In that case, reasonable approximations were made, with intuitive model testing defining the term ‘reasonable.’

Also included in the simulation subsystem is a visualization of the simulation data. This simulation data is the results of processing the input from the ECU network into models in Torcs. This visualization will be included to both aid in model verification and to demonstrate correlation between simulated “real-world” data involving the car and the processed data displayed by the middleware layer visualization. Figure 2 demonstrates the interaction between the simulation layer and the ECU-CAN network.

3.2 ECU Network Subsystem

The interface between the Torcs car simulation and the microcontroller ECU network will be implemented using a data acquisition card on the host computer. This card will provide both analog and digital I/O for transmission of data between the two layers.

The ECU network subsystem consists of individual Freescale MC9S12 micro-controllers that perform each specific function within a vehicle. More complex functions, such as those that require multiple ECU interaction (stability control, ABS), are performed by the various ECUs involved under the control of the middleware computer. Individual ECUs vary by design depending on their function, but most of them consist of a connection to the CAN network, inputs from the data acquisition (DAQ) board and the corresponding output back to the DAQ. Figure 3 demonstrates the interaction between the ECU network and the other two layers. As there are many different plants that need to be created and accounted for and many different ECUs that need to be created, the first difficulty was selecting which ECUs were to be created. The ECU’s that will be included are discussed in Section 4.2.

3.3 Middleware Subsystem

The individual ECUs only interface with the middleware through the CAN bus. This interaction between the ECU network and the middleware layer can be seen in Figure 4. The middleware subsystem consists of a laptop computer which represents the view the manufacturers have over the user’s data. The board will listen in on the CAN bus and
store and process relevant information. It will act as an interface for the following:

- **Remote Diagnostics:** Upload information to vehicle manufacturers. Manufacturers would be able to interface with a vehicle, without physically opening the vehicle, and analyze ECU data to determine what is wrong with the car.
- **Vehicle Visualizations:** Act as a dashboard to display vehicle state. This will corroborate the simulation as well as the ECU network.
- **Preemptive Maintenance:** Data mines the CAN network for information, monitors trends and stores values in databases. These databases can be analyzed for problems by vehicle manufacturers. Example: Oil levels are decreasing quickly, or leaking brake fluid. Historical data gathered from other vehicles using AutoPlug can be used to predict whether errors will occur in the future.
- **ECU Firmware Upgrades:** ECU upgrade interface for remote upgrades. Software errors can be fixed by deploying new firmware. This can be done remotely (for non-critical components), or at a dealership (for critical components).

Remote diagnostics, vehicle visualizations and preemptive maintenance are all performed by analysis on the ECU network’s CAN. ECU firmware upgrades occur by flashing the individual ECUs over the CAN network. Currently, a user would have to take in their car to a dealership and replace the entire component if a software component were broken. With AutoPlug their software can be diagnosed and upgraded over the CAN protocol.

## 4. SYSTEM IMPLEMENTATION

AutoPlug does not consist of designing the electronic barebones of a car accurately. Such a design would take years to complete, and it is not necessary to our project goals. The system does need to maintain the characteristics of a modern vehicle electronic system, such as the use of the CAN protocol and individual ECU hardware. Without the use of standard real world components in our design, AutoPlug would be useless to the target audience, as manufacturers would have to redesign the whole automobile in order to take advantage of the results.

Another distinction that needs to be made is that some of the components that will be represented in the design deal with safety-critical applications. It is recognized that there are both safety and privacy concerns associated with the final implementation of such a system, but the primary focus will remain on the implementation of a functioning product, and not on privacy and security concerns.
4.1 Simulation Subsystem

The simulation layer requires a data I/O method for communicating with the ECU network. The NI DAQ board provides ample digital and analog I/O for this task. Using appropriate send/receive/pack/unpack blocks from Torcs, the simulated car can communicate through the DAQ board. The sample rate is determined by real-time target, the complexity of the models from Torcs, and the amount of ECUs interfaced with the model. A baseline sample range is set at 0.1-1 kHz, based on typical digital control loops. The method of real-time simulation is on a Linux kernel where both visualization and data processing occurs. The simulation computer is relatively fast (3 GHz, 64-bit, 2GB RAM) in order to effectively portray the complex model. It also requires empty PCI Express slots in order to facilitate the NI DAQ board.

Models obtained from the Torcs open-source code (which are written in C and C++) are extracted and placed in a separate program which connects with the DAQ board for processing the data from the ECU network and sending this data back to the ECU network. The visualization of the automobile again occur on the Torcs system as it already contains methods for this purpose. Since the models being used come from Torcs, visualization are made to be less complex.

4.2 ECU Network Subsystem

Each ECU is implemented using an HCS12 microcontroller (MC9S12C128 variant) using the OSEK turbo real-time operating system (RTOS). Most cars employ the use of a controller (MC9S12C128 variant) using the OSEK turbo real-time operating system (RTOS). Most cars employ the use of OSEK RTOS provided by Freescale. Each ECU is written contains code that handles the transferring of the upgrade over the CAN network. As there are many different ECUs that need to be created and a model from Torcs is required for each, only certain ECUs are created. The models and ECU's that are included can be seen in Table 1. The table describes the characteristics of each ECU. One characteristic of CAN is that each node can be given an identifier. This identifier sets the priority of each node as the lower the identifier, the higher the priority. Certain ECUs receive higher priority depending on their function in a car. For example, a safety-critical component such as steering would take higher priority to a passive component such as cabin temperature. The CAN priority column describes which ECUs get the highest priority to transmit in the CAN bus.

4.3 Middleware Subsystem

The middleware layer consists of a laptop computer. The computer is running Ubuntu with GTK+ installed for graphics programming in C, which is necessary to emulate the dashboard of the ECU network. In order to communicate with the CAN bus, the laptop will use a PEAK CAN-USB adapter. Using APIs supplied by PEAK, the CAN-USB adapter converts CAN bus data into a format that is easy to work with. For remote access, XMPP, an open-standards communication protocol is used. In order to implement this, Loudmouth and Ejabberd are used. Ejabberd creates an XMPP server running on the middleware laptop while Loudmouth provides C API to send and receive from the Ejabberd server.

The codebase for the middleware application is created in C and uses the following APIs: GTK+ for programming a graphical interface, PEAK-CAN for interacting with the CAN bus over USB, and standard C libraries for programming in C. C is being used as the primary language due to its compatibility with GTK+ and the PEAK-CAN API. Using these interfaces, the main program is created that runs on the middleware laptop. This program handles the processing of data that the individual ECUs cannot handle (due to processing limitations on the microcontrollers) and handles a dashboard interface to ensure the corroborations of the simulation and ECU network layers.

This program also creates an interface by which a manufacturer may diagnose issues and upgrade firmware on the ECU network. This firmware upgrade is achieved through an initial handshake with the specified ECU to verify that both devices are ready for upgrade. The middleware layer then transmits assembly instructions along with the address in memory of those instructions. As the ECU receives these messages, the ECU flashes the instructions to the specified address on the microcontrollers memory. Once the flash is complete, the code handling the upgrade jumps to the address where the upgraded firmware now sits. For diagnostics, simple algorithms are used due to AutoPlug's goal to provide a framework for accessible rather than improved diagnostics. In addition to simple diagnostics and CAN flashing, the application can keep a history of CAN data transmitted and monitor this data for trends that can help preemptively predict errors.

Due to the complexity of the aforementioned program, threading will be used to ensure that data being written on the CAN is constantly being read and to ensure that this
<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
<th>CAN Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedals</td>
<td>Gas and brake pedals, vehicle speed</td>
<td>Disc brake actuator signals</td>
<td>High</td>
</tr>
<tr>
<td>Steering</td>
<td>Steering wheel angle sensor, speed sensor, tire angle feedback</td>
<td>Tire actuator signal</td>
<td>High</td>
</tr>
<tr>
<td>Engine</td>
<td>Speed, fuel level, RPM feedback</td>
<td>Engine timings, fuel levels</td>
<td>High</td>
</tr>
<tr>
<td>Anti-Lock Brakes (ABS)</td>
<td>Speed, wheel speeds, applied brakes</td>
<td>Applied brakes</td>
<td>High</td>
</tr>
<tr>
<td>Transmission</td>
<td>Clutch, stick, car state</td>
<td>Gear changes, car state</td>
<td>Medium</td>
</tr>
<tr>
<td>Differential</td>
<td>Lock buttons, car state</td>
<td>Lock actuators</td>
<td>Medium</td>
</tr>
<tr>
<td>Cabin Control</td>
<td>Temperature controls, temperature sensor</td>
<td>Climate elements</td>
<td>Low</td>
</tr>
<tr>
<td>Accessories</td>
<td>Light and locks</td>
<td>Lights on/off, locks on/off</td>
<td>Low</td>
</tr>
<tr>
<td>Cruise Control</td>
<td>Target speed, wheel speeds, acceleration</td>
<td>cruise control acceleration</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 1: ECU Specifications

data is constantly being processed. Due to the issues associated with threading (memory handling and deadlocks) it is important that mutexes are used to ensure that deadlocks do not occur and that data is not accessed simultaneously.

5. SYSTEM PERFORMANCE

Due to the nature of AutoPlug, it is difficult to develop a concrete assessment of the current system performance. As the goals of AutoPlug are not about speed but rather stable performance, statistics are not available for the current system in terms of efficiency. Furthermore, as the current system does not yet have a successful closed loop between all three layers, efficiency testing cannot commence. The current system consists of the successful setup of an acceleration model from Torcs with the DAQ board which has been setup and compiled on a Linux PC. Through this setup input and output can be taken to and from the DAQ in the form of an analog signal. Inputs are converted into a digital signal via the DAQ and can be viewed in the simulator.

The successful running of Linux with GTK+ and PEAK-CAN drivers on the middleware laptop have allowed for the implementation of a threaded application on the middleware laptop that reads and writes from CAN and displays this data in a graphical interface. The important feature behind this program is that it is successfully able to read from the CAN while processing the data simultaneously. This ensures that while data is being processed other data is still being read from CAN. This way, no data is lost while other data is being processed.

6. RESULTS

**Simulation Subsystem.** In addition to the setup of Torcs and the DAQ on a Linux PC, the models within Torcs have been successful interfaced for the specified ECU's. This allows for the visualization and verification of the ECU controllers in the ECU testbed. Figure 6 reveals the visualization and modeling of a physical car that Torcs is able to provide. This visualization is extremely useful in demonstrating the upgrade and diagnostic capabilities of the vehicle.

**ECU Network Subsystem.** The ECU network consists of 9 microcontrollers connected through the CAN protocol. These 9 microcontrollers emulate the control functions for the following 9 ECUs: Steering, Pedals, Cruise Control, Transmission, ABS, Engine Control, Cabin Control, Accessories and Differential. This CAN protocol is equivalent to how ECUs communicate in cars today. The ECUs emulate that controllers function, so that a bug on an ECU in the testbed represents a bug on an ECU in a real car. Moreover, an ECU tested that functions normally (as indicated by both Torcs and middleware layer) indicates a real car functioning normally. Lastly, a system has been developed for deploying new firmware onto the ECU over the CAN Network. This allows manufacturers to upgrade software components of their vehicle without physically accessing them. By having such capabilities software errors can be fixed without the need for a recall. Figure 7 contains an image of the 2 ECUs from the ECU network connected through the CAN protocol.

**Middleware Subsystem.** The application created on the middleware laptop allows for a visualization of the data from the ECU’s CAN, Figure 8. By observing the visualizations from both the simulation layer and middleware layer the function of the ECU network is verified. By reading the data on the CAN, the middleware layer is also able to perform diagnostics on this data, seen in Figure 9. Additionally,
graphing of CAN data is also available for further analysis of the CAN data. Upon completion of diagnostics, the user is notified of normal or faulty functioning of the ECU being diagnosed. The firmware can then be flashed over CAN if required. Furthermore, the option of logging the CAN data has been made available in order to perform preemptive maintenance. Lastly, an Android application has been created to receive CAN data from the middleware layer over an XMPP server. This application demonstrates the ability to send this data remotely to manufacturers for remote analysis and upgrades. This also gives the manufacturers the ability to verify upgrades with the user.

**Overall System.** Through the collaboration of these 3 systems, the following scenario can be performed with each ECU. In this example we will look at the cruise control ECU. If a user were to start their car and notice cruise control were working incorrectly, they would normally have to take their car to a dealership and the dealer would replace the entire ECU (even if the error were software related) incurring cost to both the user and manufacturer. If AutoPlug were implemented, the manufacturer can remotely run diagnostics on the car and observe the behavior in Torcs as well as in the middleware laptop seen in Figures 9 and 10a. Both results from middleware indicate a malfunctioning cruise control as the speed is not remaining constant but fluctuating rapidly. The manufacturer can then remotely flash the faulty ECU, in this case the cruise control ECU, to the newest version as seen in Figure 11. Once the upgrade is complete the ECU can be tested once again to verify its now correct operation. This can be seen in Figures 12 and 10b. This scenario can be implemented with other systems in the car, including more complex systems, such as anti-lock brakes (Figure 10c), or simpler systems, such as cabin comfort. For passive and non-critical ECUs, the manufacturer can send the user the option to upgrade their car through their smartphone as seen in Figures 13a, 13b and 13c. AutoPlug has successfully provided a framework by which manufacturers can more efficiently manage the software of their vehicles.

**7. FUTURE WORK**

Although AutoPlug has a complex 9 node network, the modern car can contain more than 100 ECUs. The addition of more ECUs raises complexity exponentially which requires more research into how the entire network can be validated when an individual ECU is upgraded. Additionally, the controller models on each ECU would have to be improved to mimic the more advance car controls rather than the simplified controls implemented in AutoPlug. Similarly, the simulation layer could be improved to contain more complex simulation models in order to perform these more complex functions. AutoPlug aims to provide a framework to discover errors rather than perform complex diagnostics. As a result, there is much room for improvement in terms of the sophistication of the diagnostics ran in the middleware layer. By performing more sophisticated diagnostics on the ECUs in the testbed, more meaningful data can be determined providing manufacturers an easy way to test and diagnose ECUs at their standards.

In addition to more complex diagnostics, diagnostics can be moved to a remote server to better simulate the real world scenario where a user is out on the road and the manufacturer is not. Currently, AutoPlug’s diagnostics and firmware upgrades occur on the middleware layer. By transmitting data to a manufacturer’s servers and running both upgrade and diagnostics there, application to the real world becomes more feasible. This however raises more questions concerning privacy and security.

Though most systems need to address the issue of privacy and security, AutoPlug was merely a framework for further research into such questions. Privacy concerns can range anywhere from how the manufacturers validate remote upgrades to how much can manufacturers find out about users
from the data they receive remotely. Additionally, the introduction of remote flashing and easier access to diagnostics may make it possible for malicious attacks on these systems. For example, if a manufacturer can flash a user’s ECU with the correct software, what is stopping someone from doing the opposite? These issues need to be addressed before AutoPlug can become a viable solution to a costly problem.

8. CONCLUSION

Remote diagnostics is an important step in the evolution of vehicle repair. The current paradigm consists of the user detecting unusual behavior, taking the vehicle in to the mechanic, being quoted an amount for the repair and finally paying the mechanic for work done. By implementing a system of remote diagnostics, the initial trip to the mechanic may be unnecessary, and the user is better equipped to make an informed decision on whether repairs need to be done and how much they would cost. Even more useful is the ability for car manufacturers to accurately detect which cars are affected by a problem in the event of a recall. As it stands now, automotive companies frequently recall more cars than they need to because they have no way of knowing which cars happen to be affected by a problem. This results in large costs in unnecessary trips to a dealership.

Preemptive maintenance goes hand in hand with remote diagnostics. Major vehicle faults are sometimes due to negligence and improper maintenance at the hands of the user, be it due to ignorance or other causes. By allowing car manufacturers to gain access to the electronic data from the ECUs, the manufacturer can alert the user about a problem before it becomes a much larger issue.

Firmware upgrading is also an important feature of AutoPlug, allowing car manufacturers to deploy new firmware for ECUs without the need to replace them. The current method of repairing electronic control units consists of replacing them with a new one. Mechanics simply have no other way to do so [8]. This costs car manufacturers and users large amount of money due to unnecessary replacement parts.

AutoPlug provides a framework for car manufacturer to remotely diagnose and correct ECU software errors. The large increase in electronic and software functions in vehicles has led to a need for manufacturers to efficiently address these problems without having to resort to traditional mechanical techniques (test and replace). AutoPlug provides a better and more cost effective method for dealing with software errors, giving the manufacturer better options at handling large scale recalls as well as isolated software errors.
9. ETHICS

As stated in Section 7, privacy and security concerns, as well as other ethical concerns, are outside the scope of AutoPlug’s goals. Privacy issues that arise are how private the information that is sent to the manufacturers is or how can the information being sent be exploited. Additionally, there can be concerns about how the current implementation can be attacked from malicious parties. Clearly, there are many ethical factors that need to be considered in a safety critical system such as a car thus making AutoPlug far from risk-free. However, these issues are issues for future research and are even a major concern in vehicles today.

10. REFERENCES

Figure 5: ECU Network Technical Model

(a) ECU Network Subsystem

(b) ECU Design Model

(c) ECU Network Specification