

Guided Integrated Remote and Workshop Troubleshooting of Heavy Trucks

Håkan Warnquist, Mattias Nyberg, and Jonas Biteus
Scania CV AB

ABSTRACT

When a truck or bus suffers from a breakdown it is important that the vehicle comes back on the road as soon as possible. In this paper we present a prototype diagnostic decision support system capable of automatically identifying possible causes of a failure and propose recommended actions on how to get the vehicle back on the road as cost efficiently as possible.

This troubleshooting system is novel in the way it integrates the remote diagnosis with the workshop diagnosis when providing recommendations. To achieve this integration, a novel planning algorithm has been developed that enables the troubleshooting system to guide the different users (driver, help-desk operator, and mechanic) through the entire troubleshooting process.

In this paper we formulate the problem of integrated remote and workshop troubleshooting and present a working prototype that has been implemented to demonstrate all parts of the troubleshooting system.

CITATION: Warnquist, H., Nyberg, M., and Biteus, J., "Guided Integrated Remote and Workshop Troubleshooting of Heavy Trucks," *SAE Int. J. Commer. Veh.* 7(1):2014, doi:10.4271/2014-01-0284.

INTRODUCTION

Trucks and buses are vehicles that are used for commercial applications. When a breakdown occurs, it is crucial that the vehicle comes back online as soon as possible. Transportation companies have small margins so all improvements they can have in vehicle uptime and costs for repair and maintenance are important.

Typically in case of a breakdown, the driver contacts a help-desk or a workshop for assistance by phone. Either a reservation is made at a nearby workshop and the vehicle is taken there, or an assistance vehicle is sent out for roadside repair. These decisions are made based on information provided by the driver and, if available from the vehicle, diagnostic information in the form of diagnostic trouble codes (DTCs) [1, 2, 3]. However, it is not until a mechanic is at the vehicle that more powerful diagnostic tests are made.

With the possibility of performing certain diagnostic tests remotely, the troubleshooting can become faster and more cost efficient. For example, if the fault is discovered to be less severe without risks of further damage to the vehicle, it is possible to continue to use the vehicle until a more suitable time for repairs arrives.

However, selecting tests to run and interpreting their results is non-trivial. We present a new system for cost-efficient guided off-board troubleshooting designed to be used both when the vehicle is in the workshop and on the road. The troubleshooting system provides decision support during the entire troubleshooting process, which begins at the moment a problem has been discovered on the road and ends when the problem is solved and the vehicle can leave the workshop. It supports help-desk operators and mechanics by listing possible problem causes and giving recommendations on how to proceed in the troubleshooting - which workshops to go to, which tests to run, and which repairs to make. A prototype implementation of the entire troubleshooting system has been created. It runs on back-office servers which communicate remotely with separate light-weight applications for the users. The implementation demonstrates the concept of guided integrated remote and workshop troubleshooting on an application example of engine temperature control and instrument cluster. [Figure 1](#) shows the part of the prototype that has been installed in the vehicle for executing remote diagnostic tests.



Figure 1. The prototype application for executing remote diagnostic tests. With the in-vehicle display to the right the driver can communicate with the help-desk when tests are executed.

The troubleshooting system uses Bayesian networks to model probabilistic dependencies between faults and observations [4]. A decision-theoretic planner is used to make cost efficient recommendations.

Several decision support systems for vehicle troubleshooting based on Bayesian networks have been proposed in the literature [5, 6, 7]. There are also decision support systems based on other techniques such as case based reasoning [8, 9, 10] and neural networks [11, 12]. None of the previous approaches considers the integration of remote and workshop diagnosis when making decisions. For the presented troubleshooting system, a modeling framework has been developed that captures the costs of both downtime and repair. Also developed is a novel planning algorithm that can consider these costs when computing cost-efficient recommendations. The troubleshooting system can provide the different users (driver, help-desk operator, and mechanic) with explanations of probable problem causes and recommend actions to proceed in the troubleshooting process.

The outline of the paper is as follows. First we present the application example used in the prototype implementation. Then we formulate the problem of creating a troubleshooting system for guided integrated remote and workshop troubleshooting. Next we present the solution where we go through the architecture of the prototype implementation, as well as the models and algorithms used.

APPLICATION EXAMPLE

To demonstrate the troubleshooting system, we have modeled a small part of the vehicle using the modeling framework presented in this paper. This model serves as an application example and it includes the parts of the vehicle that are responsible for control and monitoring of the engine temperature. An overview of the application example is shown in Figure 2. The modeled components are the coolant temperature sensor, the thermostat, the engine control unit (EMS), the coordinator control unit (COO), the instrument cluster (ICL), the red CAN bus (used by EMS), and the yellow CAN bus (used by the ICL). For each of these components we

have modeled various electrical and mechanical faults that can cause the coolant temperature to appear stuck low. We have selected this example because it is easy to set up and implement faults on it.

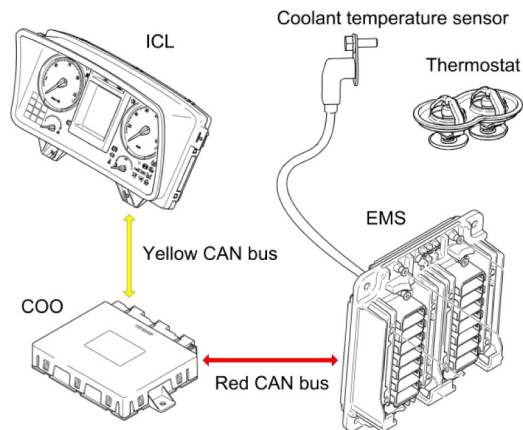


Figure 2. The components included in the application example.

PROBLEM FORMULATION

The troubleshooting system shall solve vehicle problems by sequentially recommending troubleshooting actions until the cause of the problem has been remedied. When following the recommendations, the expected cost of solving the vehicle problem shall be small. The actions that can be recommended are one of five types:

- *Reservation actions* that make new reservations at specific workshops.
- *Transport actions* that take the vehicle to the next planned workshop visit either by driving or towing the vehicle there.
- *Observing actions* that gain information by either asking a question or performing a diagnostic test (remotely or in the workshop).
- *Repair actions* that replace or repair specific components.
- The *end session action* that ends the troubleshooting session.

The intended users of the troubleshooting system are the following:

- The *help-desk operator* who uses the troubleshooting system to get recommendations of remote observing actions to execute and reservations to make. The help-desk operator also uses the troubleshooting system to input vehicle problem descriptions and other observations made by the driver.
- The *workshop mechanic* who uses the troubleshooting system to get recommendations of workshop observing actions and repairs to make. The mechanic also uses the troubleshooting system to feed back action results.
- The *driver* who can report problems and interact during remote observing actions that require the driver's input and cooperation.

Figure 3 shows the intended work flow when using the troubleshooting system. The troubleshooting session starts when the driver reports a vehicle problem to the help-desk operator. Then the troubleshooting system provides the help-desk operator with explanations of possible causes to the problem and gives recommendations on how to proceed the troubleshooting. The help-desk operator accepts the recommendation and feeds back action results that cannot be entered automatically into the troubleshooting system. Once the vehicle enters the workshop the mechanic replaces the help-desk operator as the main user of the system and continues to carry out the recommendations given until the problem is verified to be solved.

The cost of performing an action depends on the time it takes (e.g. mechanic wages and downtime related costs) and the resources consumed (e.g. spare parts and oils). The recommendations should be such that the expected total cost of solving the vehicle problem is *small*. Ideally we would want this cost to be minimal, but the additional computational effort required to achieve optimality is in conflict with the following requirements, that we have on the troubleshooting system in order for it to be useful in an industrial application:

- **Responsive.** The users must feel that the troubleshooting system is responsive. According to the human-computer interaction literature, a user will feel that a system is responsive if it takes no longer than 2-5 seconds to complete tasks with a complexity that is similar to our task of computing recommendations [13].
- **Accurate.** For all causes of a vehicle problem, the sequence of recommended actions must be such that they always eventually lead to the vehicle problem being solved.
- **Handling configurations.** The system must provide diagnoses and recommendations adapted for the current configuration of the vehicle.

Expected Total Cost

The quality of the recommendations that are given by the troubleshooting system is measured in the expected total cost of solving the vehicle problem. Assume that we can describe all relevant information of the vehicle and its environment in a state description and let π be the *recommendation policy*, a function that gives the recommendation for a given state. If we also assume that all recommendations are followed, then we can evaluate the expected total cost of completing a troubleshooting session using Bellman's dynamic programming equation [14]

$$V_{\pi}(s) = c(a, s) + \sum_{s'} \Pr(s'|a, s) V_{\pi}(s')$$

where s is a state, $V_{\pi}(s)$ is the expected cost of ending the troubleshooting session starting from state s , a is the action $\pi(s)$ recommended by the system in state s , $c(a, s)$ is the cost

of performing that action in state s , and $\Pr(s'|a, s)$ is the probability that state s' is reached when the action is performed in state s . For example, if the recommended action is a test, then s' is a state where we have added the information of the test result to s and $\Pr(s'|a, s)$ is the probability of having that particular test result in state s .

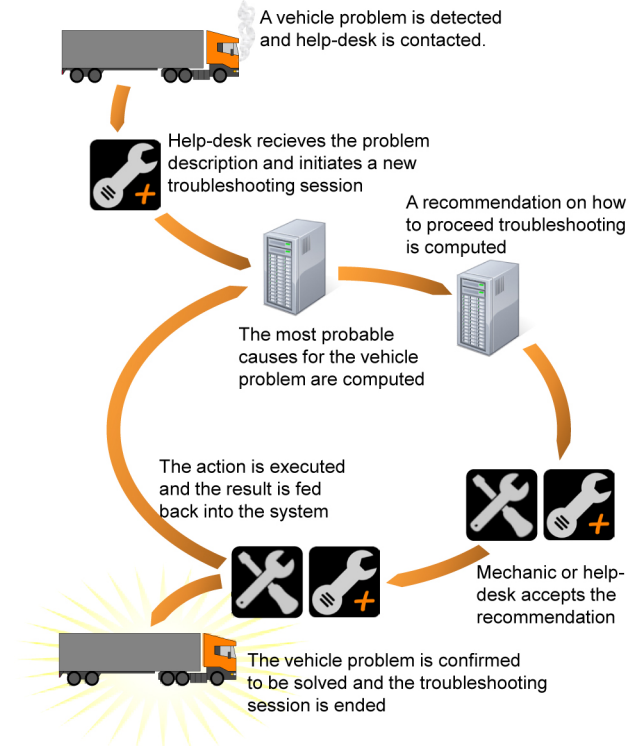


Figure 3. The work flow using the troubleshooting system.

If the troubleshooting system satisfies the accuracy requirement, i.e. it always produces a sequence of recommended actions such that the vehicle problem is eventually solved, then the expected total cost $V_{\pi}(s)$ is finite for every reachable state. Ideally we want to make the recommendations such that the expected total cost becomes minimal:

$$V^*(s) = \min_a c(a, s) + \sum_{s'} \Pr(s'|a, s) V^*(s').$$

where the optimal recommendation policy π^* is

$$\pi^*(s) = \operatorname{argmin}_a c(a, s) + \sum_{s'} \Pr(s'|a, s) V^*(s').$$

The difficulty of finding the optimal recommendation policy grows with the size of the reachable state space. If the state space is large, it is not possible to solve the equations above without violating the responsiveness requirement. However, the optimal expected cost can be used to empirically evaluate the quality of the recommendations for smaller problem instances.

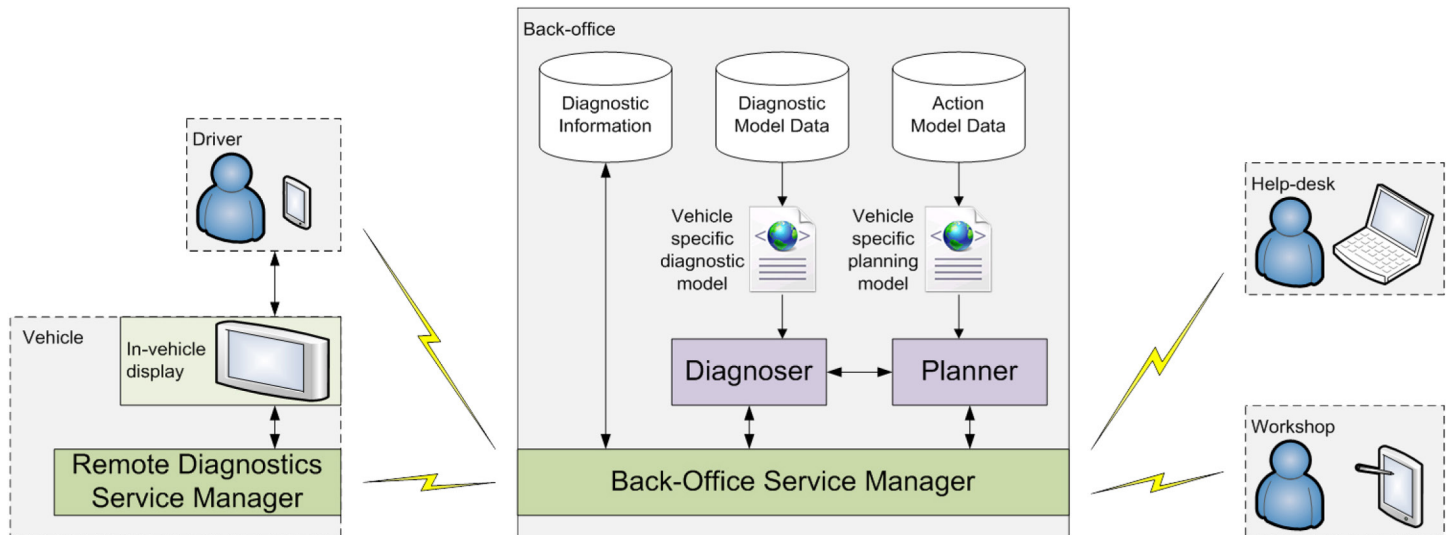


Figure 4. The architecture of the troubleshooting system.

SOLUTION

When presenting the solution we will begin by describing the architecture used in the prototype troubleshooting system and the applications created for each user. Then we will present the models and the algorithm used to produce recommendations.

Architecture

Figure 4 shows an overview of the architecture of the troubleshooting system. Most of the functionality of the troubleshooting system is placed back-office with which the users interact through their own clients.

In the back-office, a software module called the *diagnoser* is responsible for computing the probabilities of component faults and action outcomes. Another software module called the *planner* is responsible for making the recommendations. The placement back-office allows for easy access to up-to-date model data.

The *back-office service manager* provides all the services needed by the clients. It can retrieve and store diagnostic information (DTCs, reported problems, etc.), make requests to the diagnoser for fault probabilities, and make requests to the planner for recommendations and projected expected repair costs and times. The models used by the diagnoser and the planner (the *diagnostic model* and the *planning model*) are compiled together into XML-files when a troubleshooting session is started so that the models match the configuration of the current vehicle.

The users have light-weight client applications that require Internet access with the back-office to function. On the vehicle there is a *remote diagnostics service manager* responsible for sending diagnostic information to the back-office and carrying out remote diagnostic tests.

The following services are provided by the back-office application:

- *get Vehicle Information (vehicle)* Retrieves static information of a specific vehicle such as vehicle specification, owner, and registration number.
- *get Vehicle State (vehicle)* Retrieves dynamic information of a specific vehicle such as current mileage, position, DTCs, known test results and reported problems. The information is obtained from its storage back-office which is regularly updated through remote communication with the vehicle in a separate process.
- *start Troubleshooting Session (vehicle, problem)* Starts a new troubleshooting session for the specified vehicle and vehicle problem.
- *end Troubleshooting Session (session)* Ends the troubleshooting session.
- *get Possible Observations (vehicle, subsystem)* Retrieves a list of the possible observations that can be reported for the specified subsystem of the specified vehicle.
- *add Observation (vehicle, observation)* Adds an observation such as a problem report or test result to the diagnostic information of the specified vehicle.
- *remove Observation (vehicle, observation)* Removes an erroneous observation from the specified vehicle.
- *compute Fault Probabilities (session, threshold)* Lists the marginalized fault probabilities of all components related to the current vehicle problem of the session for which the probability of a fault given the current diagnostic information is greater than the threshold.
- *compute Fault Mode Probabilities (vehicle, component)* Lists the marginalized probabilities for each possible fault on the specified component given the current diagnostic information.
- *compute Next Recommendation (session)* Computes and returns the action recommended by the planner for