Project Progress Report

A SOA Framework for In-Vehicle Software
I. Overview

Our design and implementation goal is to build an in-vehicle service architecture using an Enterprise Service Bus framework. The following diagram depicts our initial overview diagram of the project that was included in our initial proposal.

Our goal is to provide a software architecture using Enterprise Service Busses to manage cross-cutting concerns such as discoverability, routing, AAA (authentication, authorization, and accounting), policy injection, privacy & security, and service orchestration.
Architectural Spike Selection

For the purpose of this course project we have decided to focus on modeling and implementing a prototype of a two-way error telemetry scenario between a car and car manufacturer, with remote software updates potentially sent to the car. Our other initial scenarios (a location-based service infrastructure and vehicle-to-vehicle ESBs) will be considered later if time allows in order to minimize our time constraint risks.

The remaining sections of this document present our Use Cases, Activity Diagrams, and initial Deployment diagram for this two-way telemetry scenario in response to errors.

II. Use Cases

The following use cases demonstrate a two-way telemetry scenario between an automobile and its manufacturer to monitor operational safety and trigger automated maintenance updates in response to high risk errors. Low and medium risk errors are also managed by our system, but can be automatically handled by our system.

High-Level Overview (From our proposal)

Data from the automobile's various domains (body, power train, chassis, infotainment center, etc.) is collected by a Data/Error Collecting Unit inside the car. In case of a serious error this data is sent as soon as possible to the car manufacturer for analysis:

1. If the problem can be resolved by a remote software update the car manufacturer sends a patch to fix the problem.
2. If the problem cannot be resolved remotely, the car manufacturer sends a message to the car's display, communicating the problem to the driver.
3. (Optional) Car Manufacturer may also send a file containing the problem analysis results to the car owner's email and preferred local mechanic to jumpstart the process of scheduling an appointment to bring the car in for repair.

For less serious errors, when the car is parked near the home it can send cumulative data to the Car Manufacturer on a regular basis through wireless Internet connections. The Car Manufacturer may analyze the data and suggest maintenance to the car owner if the brakes need adjustment or the wheels are unbalanced to optimize the car's handling.

The following diagram and specific use cases expand on the above scenario.
Use Case Diagram 1. Two-way Telemetry Scenario for High Risk Errors

In the diagram above, a sensor component in the automobile is modeled as an external actor interacting with our error monitoring system.

Main Flow:

1. Automobile sensor components signal an error in their functioning.
2. System analyzes risk factor of error.
3. In case of a high risk error requiring immediate response, System contacts Car Manufacturer.
4. Car Manufacturer, upon receiving the error, analyzes it and sends an update if one is available.
5. System notifies Car Manufacturer and Driver of change success/failure.

The next section outlines the main interactions with external entities for the above flow in further detail.
1. Use Case: Error Detection

**Description**: This use case describes the actions for the System to detect errors in the functioning of the car.

**Actors**: Automobile Sensors

**Pre-conditions**: Car is running and system is monitoring for errors.

**Trigger**: Automobile Sensors notify the system of an error, OR, the System itself identifies an error in an Automobile Sensor component or in the aggregate functioning and interaction of multiple components, rather than being directly notified of an error by a sensor component.

**Main Flow**:
1. System analyzes risk factor of error.
2. System identifies a mitigation strategy for the error.

**Alternate Flows**:
- System cannot identify a mitigation strategy so a default strategy is chosen.

**Post-conditions**: System is now aware of an error in a particular Sensor component.

**Success Conditions**: The error has been detected, analyzed for its risk factor and a mitigation strategy chosen by the System.

**Related Information**:
- Some automobile sensor components may contain their own basic error indication logic. For sensor components that do not, the System itself may provide some error detection logic for it.

2. Use Case: Update Search & Retrieval

**Description**: This use case describes the actions for the System to search for patches/updates with the Car Manufacturer in response to errors detected.

**Actors**: Car Manufacturer

**Pre-conditions**: Car is running and a high-risk error has been detected.

**Trigger**: System has become aware of a high-risk error.

**Main Flow**:
1. System initiates a telecom connection with the Car Manufacturer.
2. Car Manufacturer responds by checking to see if there is an update.
3. Car Manufacturer sends a matching update to the System.

Alternate Flows:
- If Car Manufacturer cannot be reached, System initiates another mitigation strategy (such as notifying the Driver to stop the car immediately).
- If no matching update exists, Car Manufacturer may contact 3rd party emergency roadside assistance with the location of the vehicle if the risk renders it too dangerous to drive.
- If the vehicle remains safe to drive, Car Manufacturer sends a message to the System that the Driver must be notified and maintenance scheduled in the near future.

Post-conditions: System has received instructions from the Car Manufacturer on how to respond to the error or can initiate a default error response strategy if the Manufacturer is unreachable.

Success Conditions: A response to the error has been determined.

Related Information:
- Some means of mobile telecom connection is assumed to exist in the Car.
- The Car Manufacturer’s repertoire of responses to errors may change over time, so this use case and related aspects of the System should be flexible to many operational alternatives.

3. Use Case: Update Status Notification

Description: This use case describes the actions for the System to notify the Driver and Car Manufacturer of an update.

Actors: Driver, Car Manufacturer

Pre-conditions: Car is running and an update has been downloaded from the Car Manufacturer.

Trigger: An update is ready for installation.

Main Flow:
1. System notifies Driver that a priority update must be installed, asks Driver to stop the car.
2. Driver stops the car.
3. System attempts to install the update and verifies the proper functioning of the car.
4. System notifies the Driver whether the installation succeeded/failed.
5. System notifies the Car Manufacturer whether the installation succeeded/failed.
Alternate Flows:
- Driver does not stop the car, dies in a horrible accident out of negligence and the Manufacturer is free of any legal liability.
- Stopping the car is not necessary for the update to be installed so the Driver is not notified.
- The update’s priority was reduced by the Car Manufacturer so is postponed until a more convenient time for the Driver.

Post-conditions: The update installation has succeeded or failed.

Success Conditions: System has notified the Driver and Car Manufacturer of the status of the update installation.

Related Information:
- Again, the Car Manufacturer’s repertoire of responses to errors may change over time, so this use case and related aspects of the System should be flexible to many operational alternatives.

III. Activity Diagrams

This section contains activity diagrams delineating the flows based on the above Use Cases, along with general System workflows for Error Management in a Car.
Activity Diagram 1. Main Error Handling workflow in the Car
Mitigation Strategy Selection Activity Diagram

This diagram further details the Mitigation Strategy Selection activity from the previous page.

Activity Diagram 2. Mitigation Strategy Selection workflow in the Car
Car Manufacturer Activity Diagram

This diagram depicts a high-level view of the Car Manufacturer’s workflow in response to being contacted by a remote car’s system. The activity in green is further detailed on the following page.

Activity Diagram 3. Error Handling workflow in the Car Manufacturer
IV. Domain Model

This section contains our domain model of an Error Response Management system that allows updates from a Car Manufacturer.
Error Domain Model

This diagram depicts a basic domain model and class framework for our two-way telemetry system. The logical layering starts with the Error Definition in red and continues clockwise.

Reference:

V. Deployment Model

Deployment Diagram

Original model by Prof. Krusger

Sample Deployment Workflow

1) S/D receives an Error event from Sensor Component Bus or Error Monitor detects an Error while polling.

2) Process Manager initiates main Error Response workflow depicted in the previous activity diagrams and:
   1. sends Error to Error Log
   2. asks Algorithms component for a matching Response

3) Algorithms component:
   1. identifies Error
   2. looks for a matching Error Response in Repository
   3. sends Response to Process Manager

4) Process Manager executes Error Response

5) If an Error Response is either not found or requires contacting the Car Manufacturer for an Update, the Update Process is initiated by the Do Update component

6) Update Process:
   1. Communication Module finds best connection strategy to contact Car Manufacturer
   2. Assuming Update is received the Do Update component stores it, backs up the current state of the Algorithms component, then sends new Update to Algorithms component
   3. Algorithms component installs Update, validates that there are no harmful interactions
   4. If harmful interactions exist, Do Update restores saved Algorithms state and notifies the Car Manufacturer and Driver that the Update failed and recommends a default Mitigation Strategy to be taken
   5. If no harmful interactions exist, Algorithms component installs and Process Manager executes the Update
VI. Current Status & Next Steps

Current Status:

This document reflects that we have completed initial modeling of domain and class organization, activity workflows and have a high-level understanding of our deployment structure’s workflow.

Our next step is to begin using the following implementation tools:

- The Process Manager/Execution Engine will be implemented in BPEL
- The Algorithms Component will probably use the Specification pattern including a basic Registry of Error to Error-Response mappings.

We will begin using these to prototype some simple interactions between the Process Manager and Algorithms component based on the sample workflow described in the Deployment section.

Plan until project completion:

Week 1:

Complete class diagrams, MSCs & prototype interactions on a ServiceMix ESB using a BPEL Execution Engine and Algorithms Component that manages storing and validating sensor error response rules.

Week 2:

Add sensor interaction detection and response rules to the Algorithms component. Model and implement the remaining components and focus on communication with the Car Manufacturer and developing update logic.

Week 3:

Complete implementation of the System, test.