Hochschule Darmstadt
Fachbereich Informatik

Master Projekt Systementwicklung SS2010

OBD-II - Komponente für das ICM Framework

Martin Dederer, Sven Eisenhauer

Juli 2010
betreut von Prof. Dr. Wietzke und Prof. Dr. Hergenröther

1 Zielsetzung

Aktuelle Kraftfahrzeuge verfügen über eine sog. On-Board-Diagnose-(OBD)-Schnittstelle, mittels derer der Zugriff auf Daten des Fahrzeugs möglich ist. Die Hauptanwendung dieser Schnittstelle liegt in der Auswertung von Diagnoseinformationen bei der Fehlersuche und Reparatur am Fahrzeug. Zusätzlich lassen sich über diese Schnittstelle auch einige Daten über den aktuellen Zustand des Fahrzeugs, wie beispielsweise die Motordrehzahl, abfragen.

Bei der Verbindung eines Fahrzeugs mit einem Rechnersystem kommt in den meisten Fällen ein Adapter zum Einsatz, der zum Rechner hin eine serielle Datenschnittstelle aufweist. Über diese Verbindung lassen sich Fahrzeugdaten von einem Rechnersystem auslesen.

Die Zielsetzung dieses Projekts besteht in der Anbindung eines Fahrzeugs an einen Rechner. Zum Auslesen der Fahrzeugdaten soll eine Komponente für das ICM-Framework geschrieben werden, das auf dem Rechner laufen soll. Die Anzeige der Daten soll mittels eines digitalen Instrumentes auf dem Bildschirm des Rechners erfolgen. Auch diese Anzeige soll in das Framework integriert sein. In dieser Version beschränken wir uns auf die Darstellung der Motordrehzahl mit einem digitalen Drehzahlmesser.

2 Fahrzeug-Schnittstelle

2.1 Hardware

Die mechanische Verbindung des Diagnoseadapters erfolgt über eine genormte 16-polige Schnittstelle. Neben einer Spannungs- und Masseleitung verfügt diese Schnittstelle je nach Fahrzeugtyp über unterschiedliche Datenleitungen.

Am Markt sind unterschiedliche sog. Multiprotokoll-Adapter erhältlich, die diese Unterschiede abstrahieren. Vor der Anschaffung eines Adapters empfiehlt sich eine Recherche, ob der gewünschte Adapter mit dem Fahrzeug kompatibel ist. Für dieses Projekt fand ein Adapter vom Typ AGV4000 der Firma OBD-DIAG Verwendung. Damit konnten erfolgreich Verbindungen zu einem Forst Fiesta MK6 und einem Skoda Roomster aufgebaut werden. Der Adapter verfügt über eine USB-Schnittstelle, die unter Linux und Windows funktioniert.

2.2 Protokolle

Ebenso wie bei der mechanischen Schnittstelle existieren modellabhängige Unterschiede bei den Protokollen für die Datenübertragung. Auch diese Unterschiede abstrahiert ein Multiprotokoll-Adapter mittels eines Interpreter-Chips. Dieser setzt das fahrzeugspezifische Kommunikationsprotokoll auf einheitliches um. Beim AGV4000 kommt ein Chip der Firma ELM zum Einsatz, deren Chips auch in vielen anderen Adaptern Verwendung finden.

Das Protokoll zur Kommunikation zwischen Adapter und dem Rechner gliedert sich grob in zwei Kategorien. Zum einen existieren sog. AT-Befehle zur Konfiguration des Adapters selbst, zum anderen Befehle zur Kommunikation mit dem Fahrzeug. Der AGV4000 verfügt über eine automatische Erkennung des Fahrzeugprotokolls, die bei den getesteten Fahrzeugen die richtigen Einstellungen vornahm. Alternativ lassen sich die Konfigurationsparameter mittels der entsprechenden AT-Befehle einstellen und auslesen.

Die Abfrage von Fahrzeugdaten erfolgt über sog. OBD-II Parameter-IDs, kurz PIDs oder P-Codes. Diese sind im Standard SAE J/1979 definiert. Nicht alle Fahrzeuge unterstützen alle im Standard definierten PIDs, umgekehrt implementieren viele Hersteller zusätzliche PIDs. Der Standard ist leider nicht frei verfügbar, allerdings sind die wichtigsten PIDs bekannt. Allgemein erfolgt die Abfrage eines Fahrzeugdatums durch das Versenden des entsprechenden P-Codes an den Adapter. Dieser setzt die Anfrage auf das fahrzeugspezifische Protokoll um und versendet es an dessen Schnittstelle. Von dort wird die Anfrage über den Fahrzeug-Bus, z. B. CAN, an das zuständige Steuergerät weitergeleitet. Dieses verarbeitet die Anfrage und schickt das Resultat über den Fahrzeug-Bus zurück an die OBD-Schnittstelle. Dort empfängt der Adapter das Resultat, setzt es ggf. um und leitet es über die serielle Schnittstelle zum Rechner weiter.

Eine Anfrage setzt sich aus 4 Zeichen zusammen, wovon jeweils 2 Zeichen die hexadezimale Darstellung des Modes und des PIDs bilden. Ein abschließendes Carriage-Return-Zeichen führt die Anfrage aus. Somit lassen sich OBD-II-Anfragen auch mittels eines Terminal-Programms wie HyperTerminal unter Windows ausführen. Das Format einer OBD-II-Anfrage folgt dementsprechend der Beschreibung

<Mode><PID>\r

Der Mode legt die Art der Anfrage fest, beispielsweise erfolgt die Abfrage von aktuellen Daten über den Mode 01 und die Abfrage von Fehlercodes über den Mode 03. Für das Projekt benötigen wir ausschließlich aktuelle Daten, weshalb wir uns auf den Mode 01 beschränken.

Die PID zur Abfrage der Motordrehzahl lautet 0C. Somit ergibt sich die Zeichenkette

100C\r

zur Abfrage der Motordrehzahl. Diese ist vom Programm über die serielle Schnittstelle an den Adapter zu Versenden.

Die Antwort erfolgt ebenfalls in Form von hexadezimalen Werten. Je nach angefragtem Wert unterscheidet sich die Länge der Antwort. Sollte eine Anfrage mehrere Einzelwerte zurück liefern, so sind diese jeweils mittels eines Zeilenumbruchs getrennt. Dies kann der Fall sein, wenn mehrere Fehlercodes vorliegen. In diesem Fall wird jeder Fehlercode in einer eigenen Zeile übertragen. Im Normalfall schickt der Adapter nachdem er o. g. Anfrage erhalten hat nach 20 Millisekunden alle Daten, die er bis zu diesem Zeitpunkt vom Fahrzeug empfangen hat, weiter. Daraus ergibt sich eine Latenz von 20 Millisekunden zwischen Anfrage und Antwort im anfragenden Programm.

Eine Besonderheit der ELM-Chips liegt in der Möglichkeit durch Angabe der erwarteten Zeile in der Antwort die Wartezeit auf die Antwort des Adapters zu verkürzen. Dazu verstehen ELM-Chips einen zusätzlichen Wert im Anfrage-String:

<Mode><PID><No of Lines>\r

Der dritte Wert gibt an, nach wie vielen Zeilen der Adapter sofort den Wert weiterreichen soll und nicht auf weitere Daten warten soll.

Da ein Datum für die Motordrehzahl immer aus einer Zeile mit 4 Bytes besteht, lässt sich durch diesen Anfrage-String die Antwortzeit deutlich verkürzen:

100C1\r

Das Format einer entsprechenden Antwort lautet

<Mode><PID><A><B>\r

Wobei Mode sich hier aus dem Mode der Anfrage plus 40 hex berechnet, für Anfragemode 01 also 41. Der PID entspricht dem der Anfrage. A und B stellen zwei hexadezimale Zahlen dar, aus denen sich die Motordrehzahl nach der Formel

$rpm = ((A * 0x100) + B)/4$

berechnet.

3 Implementierung

Dieser Abschnitt beschreibt die wichtigsten Aspekte der Implementierung der Komponenten für das ICM-Framework zur Abfrage und grafischen Darstellung von OBD-II-Daten. Die Komponenten sollen auf Linux x86 und QNX x86 lauffähig sein, weshalb sich die Implementierung am POSIX-Standard orientiert und nur entsprechende System-Routinen verwendet. Die grafische Darstellung richtet sich nach dem OpenGL ES Standard. Eine Implementierung für QNX auf SH4 Prozessoren erfolgt nicht, da diese Zielplattform über keine USB-Schnittstelle zum Anschluss eines OBD-Adapters bietet.

3.1 Simulator

Bei der Entwicklung von OBD-Software erweist sich der Einsatz eines Simulators als sehr hilfreich. OBD-Simulatoren sind zum Einen am Markt als Hardware-Lösung verfügbar. Diese werden mittels OBD-Adapter mit dem Entwicklungsrechner verbunden und liefern einstellbare Werte über die OBD-II-Schnittstelle. Die Beschaffung eines solchen Simulators befindet sich in der Planung. Zum Anderen existiert ein reiner Software-Simulator als Bestandteil des Open-Source-Projekts obdgpslogger (http://icculus.org/obdgpslogger/). Dieser simuliert auf einem Pseudo-Terminal einen OBD-II-Adapter nach dem ELM-Befehlssatz. Über eine optionale grafische Oberfläche lassen sich bestimmte Werte einstellen, die der Simulator über das Pseudo-Terminal an das anfragende Programm zurück gibt. Die Software ist im Quellcode verfügbar und lässt sich mittels cmake übersetzen. Dies war leider auf den Laborrechnern nicht möglich, da einige Bibliotheken, von denen obdgpslogger abhängt nicht installiert waren. Das cmake Buildsystem liefert eine Liste der fehlenden Bibliotheken, unter anderem FLTK für die GUI und FFT. Somit erfolgte die Entwicklung auf privaten Rechnern, auf denen die notwendigen Bibliotheken installiert werden konnten, um den Simulator von obdgpslogger einsetzen zu können.

3.2 Komponente

Für die Kommunikation mit dem OBD-II-Adapter über die serielle Schnittstelle des Rechners erfolgt über ein logisches Device im ICM-Framework. Dieses stellt eine eigenständige Komponente dar, die in einer separaten Ablaufeinheit ausgeführt wird. Die Implementierung erfolgt in einer C++-Klasse, die den Vorgaben des ICM-Frameworks folgt. Maßgeblich hierbei sind neben anderen Methoden die init- und run-Methode.

Die init-Methode jeder Komponente wird einmalig beim Start des Frameworks ausgeführt. Die OBD-Komponente öffnet hier die entsprechende serielle Schnittstelle über einen einfachen Mechanismus zur automatischen Erkennung. Dieser erleichtert die Entwicklung mit dem Simulator, da dieser bei jedem Start das nächste freie Pseudo-Terminal auf einem Linux-System verwendet. Deshalb lässt sich nicht von vornherein festlegen, welche Schnittstelle von der Komponente zu verwenden ist. Ohne diesen Mechanismus müsste nach jedem Start des Simulators der Quellcode der Komponente angepasst und neu übersetzt werden. Außerdem verbirgt dieser Mechanismus die unterschiedlichen Pfade zur seriellen Schnittstelle unter Linux und QNX. Nach dem Öffnen der Schnittstelle konfiguriert die init-Methode diese entsprechend. Wichtig ist hierbei die Verwendung der Schnittstelle im Rohdaten-Modus. Nach unseren Erfahrungen betreibt Linux eine serielle Schnittstelle standardmäßig im kanonischen Modus, was zu einem Problem mit der select-Systemfunktion führt. Dies wird später genauer erläutert.

Die run-Methode wird nach der Initialisierung einer Komponente aufgerufen. In ihr findet sich die Implementierung der Funktionalität der Komponente. Hierbei handelt es sich um eine Endlos-Schleife, die nur unter bestimmten Bedingungen verlassen wird. In dieser Schleife führt die Komponente drei Aktionen aus. Sie schreibt den Anfrage-String auf die serielle Schnittstelle. Danach wartet sie ohne CPU-Zeit zu verbrauchen auf die Antwort-Daten. Im dritten Schritt verarbeitet sie die empfangenen Daten, legt sie im

Datencontainer ab und schickt eine entsprechende Nachricht über den Main Dispatcher des Frameworks.

Für das Warten ohne CPU-Zeit bietet sich auf POSIX-Systemen die Systemfunktion select an. Diese blockiert den aufrufenden Thread und kehrt entweder nach einem konfigurierbaren Timeout oder bei empfangenen Daten zurück. Hierbei trat das oben erwähnte Problem auf. Im kanonischen Modus erkennt das System nicht, dass neue Daten an der Schnittstelle anliegen und kehrt erst nach Ablauf des Timeouts zurück. Je nach Höhe des Timeouts führt dieses Verhalten zu einer sehr ruckeligen Darstellung des Drehzahlmessers. Erst bei Betrieb der seriellen Schnittstelle im Rohdaten-Modus arbeitet die select-Funktion wie erwartet. Dieses Problem tritt bei der GPS-Komponente nicht auf, dort arbeitet die serielle Schnittstelle im kanonischen Modus korrekt.

3.3 Datencontainer

Gemäß den Vorgaben des ICM-Frameworks handelt es sich bei einem Datencontainer um einen Bereich im Shared-Memory, auf den mehrere Komponente zugreifen und so Daten austauschen. Hierbei ist besonders darauf zu achten, dass die Zugriffe auf das Shared Memory über einen Mutex geschützt werden. Dies erfolgt in der Implementierung des Datencontainers, so dass dieser Synchronisierungsmechanismus vor dem Benutzer des Datencontainers verborgen wird. Die Motordrehzahl wird als Fließkommazahl im Shared Memory abgelegt. Der Zugriff auf den Datencontainer erfolgt über zwei Zugriffsklassen, die in den folgenden Abschnitten beschrieben werden.

3.3.1 Accessor-Klasse

Diese Klasse kapselt den schreibenden Zugriff durch Komponenten auf den Datencontainer. Die OBD-Komponente verwendet diese Klasse, um die aktuell empfangene Drehzahl im Datencontainer speichern.

3.3.2 Adapter-Klasse

Für den lesenden Zugriff auf den Datencontainer stellt die Adapter-Klasse eine Methode für jeden Wert im Datencontainer bereit. Somit bleibt Client-Komponenten der innere Aufbau des Datencontainers verborgen.

3.4 Digitaler Drehzahlmesser

Eine weitere Komponente übernimmt die Darstellung der Motordrehzahl auf dem Bildschirm. Die Darstellung orientiert sich an einem analogen Drehzahlmesser, wie er aus Fahrzeugen bekannt ist. Dabei kommen meist Kreisinstrumente mit Drehzeiger zum Einsatz. Der Drehwinkel des Zeiger veranschaulicht die Motordrehzahl.

Um auch hier CPU-Zeit ein zu sparen, zeichnet die Komponente nicht ständig die Anzeige neu, sondern nur wenn neue Daten vorhanden sind. Der Einfachheit halber erfolgt diese Synchronisation über den Nachrichten-Dispatcher der Komponente. Dieser blockiert auf einem binären Semaphore im Kontext der Komponente. Die Update-Nachrichten der

OBD-Komponente sind an diese Komponente adressiert. Somit der Dispatcher der Komponente bei jeder neuen OBD-Update-Nachricht geweckt. Diese Nachrichten treffen nur ein, wenn neue OBD-Daten im Datencontainer vorliegen. Nach dem Verarbeiten einer OBD-Update-Nachricht, was sich auf den Empfang beschränkt, liest die Komponente die neue Motordrehzahl über die Adapter-Klasse aus dem Datencontainer.

Mit diesem Wert berechnet sie den Drehwinkel des OpenGL-Zeigers neu und rendert den Anzeigebildschirm. Jetzt blockiert die Komponente bis zum Eintreffen einer neuen OBD-Update-Nachricht.