TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Verkehrswissenschaften „Friedrich List“ - Institut für Automobiltechnik Dresden

Lehrstuhl Kraftfahrzeugtechnik

STUDIENARBEIT

Realisierung einer Busüberwachungs-Schnittstelle für das Testautomatisierungswerkzeug ECU-TEST zum Busüberwachungswerkzeug CARMEN

Bearbeiter
Sven Richter

Geboren am
18.04.1979 in Karl-Marx-Stadt (jetzt Chemnitz)

Betreuer
Dipl.-Ing. Andreas Unger - TU-Dresden, IAD
Dipl.-Ing. Frank Günther - TraceTronic GmbH

Betreuender Hochschullehrer
Prof. Dr.-Ing. B. Bäker

Tag der Einreichung:
01.08.2008

TECHNISCHE UNIVERSITÄT DRESDEN
Fakultät Verkehrswissenschaften „Friedrich List“
Institut für Automobiltechnik Dresden (IAD)
Lehrstuhl Fahrzeugmechatronik

Aufgabenstellung für die Studienarbeit

im Studiengang:	Mechatronik
Studienrichtung:	Mikromechatronik
Mat-Nr.:	2685340
Name:	Sven Richter

Thema : **Realisierung einer Busüberwachungs-Schnittstelle für das Test-automatisierungswerkzeug ECU-TEST zum Busüberwachungswerkzeug CARMEN**

Zielstellung:
Durch den steigenden Softwareanteil in Steuergeräten von Automobilen sind dessen Tests ein wesentlicher Bestandteil der Qualitätssicherung. Die Ausführung von Testsequenzen mit dem Softwaretool ECU-TEST bildet dafür eine optimale Grundlage. Bei der Durchführung dieser Testsequenzen zeichnen verschiedene an ECU TEST angekoppelte automobilspezifische Softwarewerkzeuge Daten und Werte von Steuergeräten über der Zeit auf. Für die Auswertung des zeitlichen Verhaltens von Bussystemen, ist die Erweiterung um Funktionalitäten zur Busüberwachung notwendig.
Ziel der Arbeit ist es, eine Schnittstelle für die Busüberwachung auf Basis des Werkzeugs CARMEN in ECU-TEST zu integrieren

Schwerpunkte der Arbeit:

- Entwurf von generischen Testschritten zur Busüberwachung:
- Entwurf eines Templates für ein CARMEN-Modul zur Realisierung der Überwachungs-/Analysefunktionen.
- Konzept für die Anbindung des Werkzeugs CARMEN
- Umsetzung in ECU-TEST
- Implementierung eines CARMEN-Moduls zum Test der Schnittstelle

Betreuer:
Dipl.-Ing. A. Unger
(TU Dresden)

Frank Günther
(Tracetronic GmbH)

Ausgehändigt am: 31.03.2008

Prof. Dr.- Ing. B. Bäker

Betreuender Hochschullehrer

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie die Zitate deutlich kenntlich gemacht zu haben.

Dresden, den 01.08.2008 Sven Richter

Inhaltsverzeichnis

1 Einführung **1**

1.1 Motivation . . . 1
1.2 Präzisierung der Aufgabenstellung . . . 2
1.3 Testautomatisierungswerkzeug ECU-TEST . . . 2
1.4 Busüberwachungswerkzeug CARMEN . . . 3

2 Softwaretechnische Grundlagen **5**

2.1 Programmschnittstelle COM . . . 5
2.1.1 Die Interface Description Language . . . 5
2.1.2 Eindeutige Klassenreferenz . . . 7
2.1.3 Verarbeitung von Ereignissen . . . 7
2.1.4 Rollenverteilung bei der Kommunikation . . . 8
2.2 Programmiersprache Python . . . 10
2.2.1 wxPython . . . 11
2.2.2 Python und COM . . . 11

3 Entwurf der Schnittstelle **13**

3.1 Realisierungsvarianten . . . 13
3.2 ECU-TEST Integration . . . 16
3.2.1 Aufbau des Testautomatisierungstools . . . 16
3.2.2 Nötige Modifikationen . . . 18
3.3 Entwurf eines Moduls zur Busüberwachung . . . 19

4 praktische Umsetzung der Anbindung **22**

4.1 ECU-TEST-CARMEN-COM-Server . . . 22
4.1.1 Klasse ”PyUtilities” . . . 23
4.1.2 Klasse ”Data” . . . 23
4.1.3 Klasse ”ModulData” . . . 24
4.2 Test-Klassen und Toollib für ECU-TEST . . . 24
4.2.1 Toollib und das zugehörige Panel zur Konfiguration . . . 25
4.2.2 BUSViewer . . . 27

4.2.3 Testklasse und CARMEN-Client 28
4.2.4 Teststep-Klassen und zugehörige Konfigurationsdialoge 30
4.2.5 Testablauf 32
4.3 CARMEN-Modul 33
4.4 Schlussfolgerung 39

5 Zusammenfassung und Ausblick **40**

Literaturverzeichnis **41**

A UML Diagramme **44**
A.1 COM-Server 44
A.2 Toollib 45
A.3 Konfigurationspanel 46
A.4 Teststep-Container 47
A.5 Busviewer 49
A.6 Integration in die Regression 50

1 Einführung

1.1 Motivation

Autos im Einundzwanzigsten Jahrhundert, das heißt zeitgemäße, funktionell hochwertige, sichere und formschöne Fahrzeuge. Eine Vielzahl an Forderungen, Qualitätsparametern und sonstigen Eigenschaften müssen zusammengefaßt werden, um Automobile unterschiedlichster Anforderungen zu entwickeln. Der hohe technische Standart und der hohe Automatisierungsgrad erfordern zwingend aussagefähige und qualifizierte Tests in vielen verschiedenen Entwicklungs- und Fertigungsphasen.

In der modernen Automobilindustrie ist der Test der komplexen Systeme, welche unsere heute üblichen Fahrzeuge bilden unerlässlich. Gerade die Vielzahl an Bussystemen und Steuergeräten gestaltet die Validierung der korrekten Funktionalität problematisch. Außerdem fordern entsprechende Normen eine Kontrolle der implementierten Sicherheitsmaßnahmen und ihre Zuverlässigkeit.

Durch den Variantenreichtum an unterschiedlichen Steueraggregaten und Bussen ist nur ein automatisierter Test in der Lage, eine nahezu vollständige und optimale Überprüfung zu ermöglichen. Eine manuelle Testdurchführung ist in diesem Rahmen nicht mehr wirtschaftlich vertretbar.

Für viele typische Validierungsprobleme existieren bereits spezialisierte Anwendungen, die Randprobleme oder einzelne Bereiche von sich heraus abdecken. Um allerdings ein nahezu einheitliches und vorzugsweise vollständiges Testprotokoll zu ermöglichen, ist es notwendig, diese verschiedenen Tools zu vereinen. Gegebenenfalls kann eine Adaption der verschiedenen Testdurchführungen erforderlich werden.

1.2 Präzisierung der Aufgabenstellung

Ziel dieser Arbeit ist es, eine Schnittstelle für die Busüberwachung auf Basis des Werkzeugs CARMEN in Testautomatisierungstool ECU-TEST zu integrieren. Dabei muss auf mehrere Aspekte der Schnittstellenrealisierung eingegangen werden.

Für die Steuerung aus ECU-TEST sind mehrere Testschritte zu entwerfen. Dazu gehören zum einen Schritte zum Starten und Stoppen des Busmonitoring und Konfigurieren der Überwachungsfunktionen. Natürlich sind auch Testschritte zum Auswerten der Überwachungsergebnisse zu konzipieren. Da eine Busüberwachung nur mit einem angepassten Carmen Modul möglich ist, ist auch der Entwurf eines entsprechenden Moduls zur Realisierung der Überwachungs-/Analysefunktionen Inhalt dieser Arbeit.

Ein weiterer Inhaltsaspekt ist die Beschreibung der Umsetzung in ECU-TEST. Sie besteht aus mehreren Teilen. Die Erweiterungen für Testschritteinfüge-/parametrierungs Funktionalitäten und Erstellung der ToolLib für die Kommunikation mit CARMEN gehören ebenso dazu, wie die Integration in die Prüfumgebungskonfiguration zur Einrichtung der ToolLib. Selbstverständlich wird dazu ebenso das Konzept zur Anbindung des Buswerkzeuges CARMEN behandelt.

Abschließend soll die Implementierung eines einfachen CARMEN-Moduls beschrieben werden. Dies ist nötig, um die implementierte Schnittstelle zu testen. Dabei wird auf die spezifischen Eigenschaften eingegangen, die solch ein Modul besitzen muss, um in ECU-TEST angesprochen werden zu können.

1.3 Testautomatisierungswerkzeug ECU-TEST

”ECU-TEST ist die Testautomatisierungs-Software für die Validierung eingebetteter Systeme im Automotive-Umfeld. ECU-TEST findet Anwendung beim Design, der Realisierung, der Durchführung und der Auswertung von Tests einschließlich der Generierung von intuitiven Testreports.”[Gmb]

Daher bietet ECU-TEST (Abbildung 1.1) Testklassen für die im Fahrzeug vorherrschenden Bus-Systeme wie CAN und FlexRay und die Unterstützung für Simulationswerkzeuge wie Matlab-Simulink und CANoe. Durch die modulare Architektur lassen sich auch problemlos weitere Systeme unterstützen.

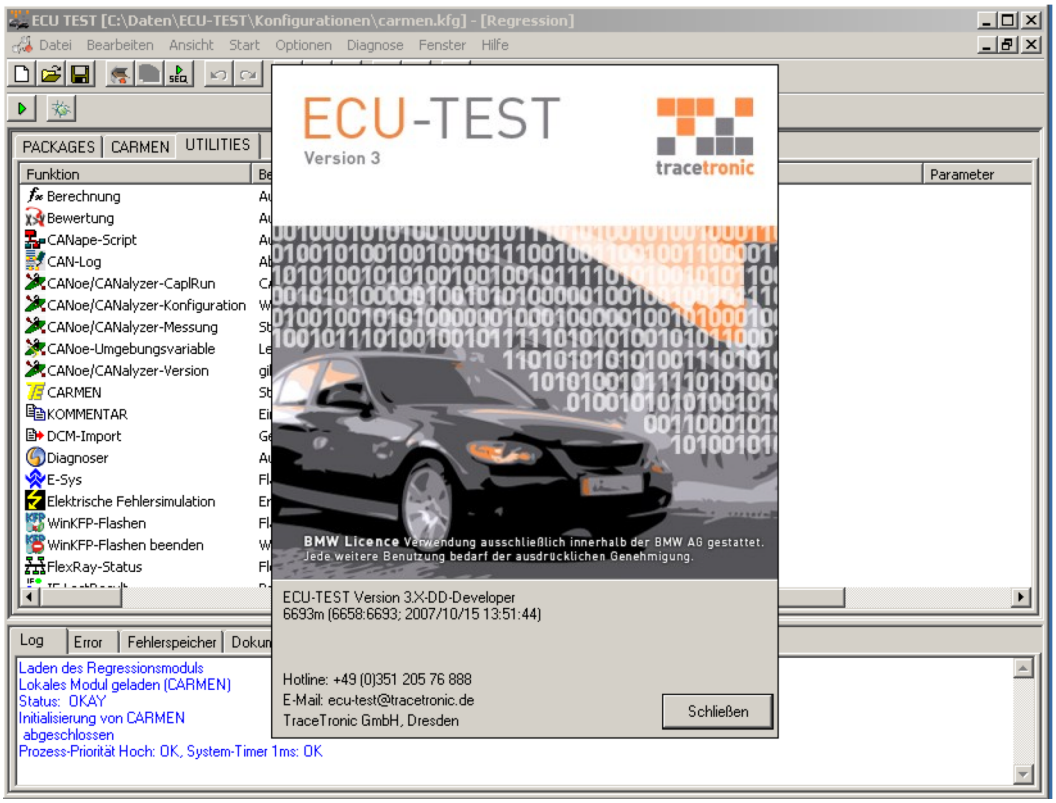

Abbildung 1.1: ECU-TEST Anwendung

In ECU-TEST werden Versuche durch sequentielle Abfolge verschiedener Testschritte generiert. Natürlich ist es auch möglich, Schleifen und bedingte Anweisungen zu erstellen um komplexe Szenarien zu realisieren. Die Testsequenzen werden in einer grafischen Umgebung durch einfaches Drag und Drop erstellt. Dadurch ist eine vielfältige Funktionalität und einfache Bedienung, die nur minimale Einarbeitung benötigt, gewährleistet.

Außerdem enthält ECU-TEST zusätzlich einen Applikationsserver, womit es möglich wird Softwaretools und Schnittstellen auf entfernten Prüfstandsrechnern über das lokale Netzwerk anzusprechen.

1.4 Busüberwachungswerkzeug CARMEN

”CARMEN steht für Car Measurement Environment. CARMEN ist ein offenes, erweiterbares Werkzeug (genauer: eine Sammlung von Werkzeugen), um Messungen am Bussystem

eines Fahrzeugs durchzuführen. Auch die Stimulation wird unterstützt, so dass auch Simulationsaufgaben abgedeckt werden."[Ing]

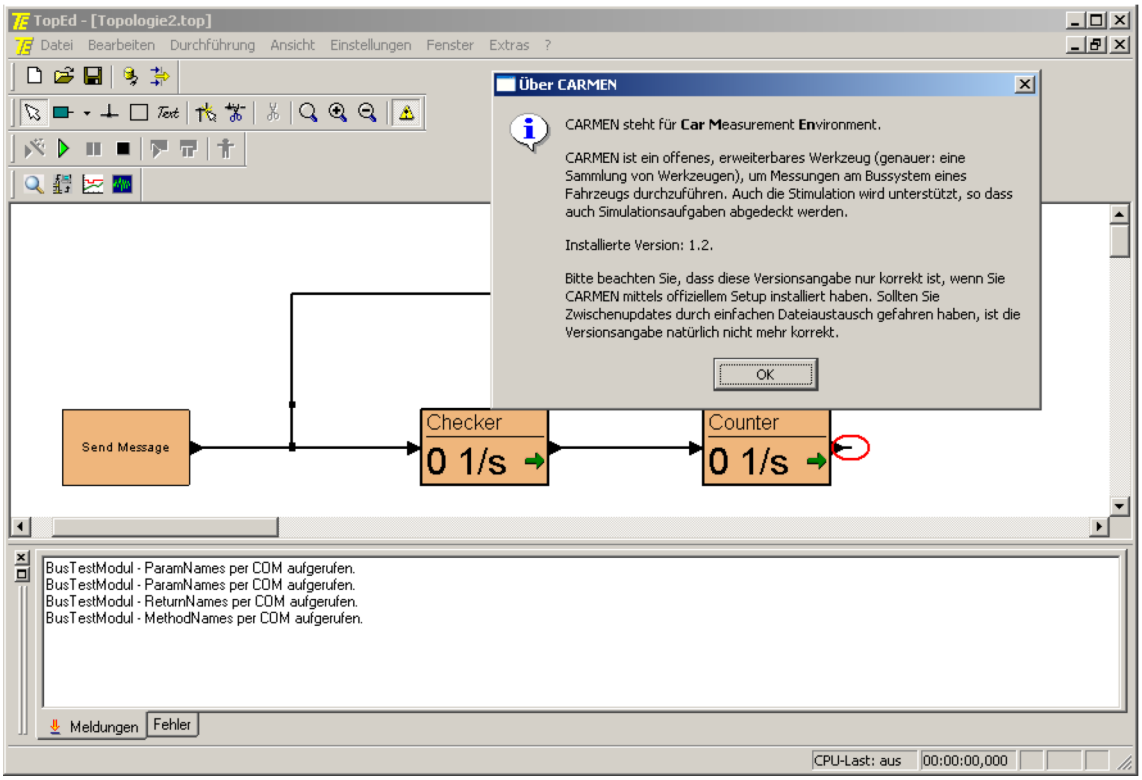

Abbildung 1.2: CARMEN - TopEd Anwendung

Mit CARMEN-TopEd (1.2) kann man, ähnlich Simulink, die verschiedenen Module in einem Netzplan anordnen. Standartmäßig sind zahlreiche Module wie ein Paketzähler, Paketlogger, Filter und so weiter vorhanden. Diese werden so angeordnet, dass entsprechend ihrer Funktion die Ein- und Ausgänge verbunden werden können.

Um spezielle Funktionen zu realisieren, ist es möglich CARMEN durch eigene Module zu erweitern. Diese können in C++ oder Visual Basic implementiert werden. Außerdem besitzt CARMEN eine COM-Schnittstelle zur Automatisierung, womit grundlegende Programmfunktionen, wie zum Beispiel "Konfiguration laden", "Test starten" et cetera, aufgerufen werden können.

Vom Benutzer erzeugte Module müssen vorgegebene Methoden integrieren und arbeiten eventgesteuert. So wird bei jedem Paket eine entsprechende Methode aufgerufen, die dieses anschließend verarbeitet. Des Weiteren besteht die Möglichkeit Timer zu definieren, welche dann gemäß ihrem Intervall eine Methode aufrufen.

2 Softwaretechnische Grundlagen

2.1 Programmschnittstelle COM

Die Abkürzung COM steht für Component Object Model und ist Microsofts Industriestandard, um bei Windows Plattformen verteilte Objekte handhaben zu können. Dabei ermöglicht die Erweiterung DCOM dies auch über das Netzwerk. Bei diesem Projekt wurde COM genutzt, um die Kommunikation zwischen den Programmen zu realisieren und deshalb sollen in diesem Kapitel kurz die Funktionsprinzipien und Konzepte beschrieben werden. Die Common Object Request Broker Architecture, kurz CORBA, beschreibt ähnliche Prinzipien, wurde allerdings von einem Konsortium, bestehend aus mehr als 800 Firmen der Computer-Industrie erarbeitet. In COM findet man diese Konzepte ähnlich oder teilweise sogar identisch realisiert.

2.1.1 Die Interface Description Language

Da die einzelnen Programme, die über COM (Abbildung 2.1) kommunizieren in den unterschiedlichsten Programmiersprachen erstellt sein können, muss ein einheitliches Format zum Austausch der Schnittstellendefinition gefunden werden. Die Interface Description Language, kurz IDL, wird genutzt um eine Programmiersprachen unabhängige Interface Beschreibung vorzunehmen. IDL wird ausser bei COM und CORBA auch in Sun's ONC RPC , The Open Group's Distributed Computing Environment, IBM's System Object Model, Facebook's Thrift und WSDL for Web services verwendet.

Der Syntax von IDL weist trotz der erforderlichen Sprachneutralität eine starke Ähnlichkeiten mit C++ auf. Dies ist natürlich auch beabsichtigt, da C++ in der Softwarebranche eine der dominierenden Sprachen ist. "Der erste augenscheinliche Unterschied ist der, dass man mit IDL Schnittstellen (IDL-Schlüsselwort interface) und keine Klassen beschreibt"[Loo01] Damit wird auch wirklich nur die Schnittstelle beschrieben, eine Realisierung der dafür zuständigen Klassen findet zu einem späteren Zeitpunkt statt. Zu

einem IDL Schlüsselwort gehört ein Reihe von IDL Attributen, die in eckigen Klammern aufgeführt sind und davor notiert werden.

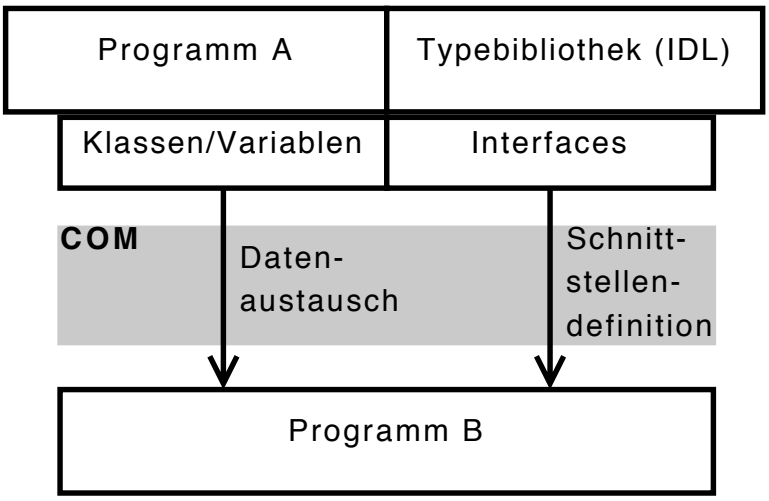

Abbildung 2.1: COM Kommunikation

Bei der Definition von Methoden ist die Ähnlichkeit zur C++ Syntax ebenfalls deutlich zu erkennen. Allerdings werden einige Schwachstellen von C++ bei der Parameterdefinition beseitigt. Zum Beispiel wird für den Variablentyp int in C nicht festgelegt, wie viel Speicher reserviert werden muss. Je nach Compiler und Rechner kann dies unterschiedlich ausfallen. Solange man auf einem einzelnen Rechner agiert ist das völlig ausreichend, allerdings bei der Verwendung von verteilten Objekten über COM sorgt dies für einige Probleme.

Deshalb verwendet man in IDL einige wenige Basisdatentypen die in ihrer Größe und Vorzeichen Interpretation konkret definiert sind. Ausserdem ist bei jedem Parameter ein so genanntes Richtungsattribut, wie in oder out, mit anzugeben. Diese Kennzeichnung legt fest, in welcher Richtung zwischen Client und Objekt die Werte dieser Parameter zu transferieren sind.

”Das COM-Laufzeitsystem kann mit IDL-Dateien selbst nichts anfangen, wir benötigen eine Darstellung, die auf einem binären Format beruht.”[Loo01] Für Windows Umgebungen ist diese Binärdarstellung die Typeblibliothek. Damit zu einer IDL-Datei mit Hilfe des MIDL-Compilers eine korrespondierende Typebibliothek erzeugt werden kann, muss die IDL Datei zusätzlich zur Interface-Deklaration einen Deklarationsabschnitt ”library” enthalten. Deren Attributliste enthält die Attribute „uuid“, „helpstring“ und „version“. Dabei ist besonderes die Bedeutung der UUID hervorzuheben und wird daher im nächsten Abschnitt näher erläutert.

2.1.2 Eindeutige Klassenreferenz

Die GUID, globally unique identifier, ist eine 128 Bit grosse Integer Zahl die weltweit eindeutig ist. In Texten verwendet man für GUIDs stets eine spezielle Darstellung (32 Hexadezimalziffern mit 4 Bindestrichen) wie zum Beispiel:

```
BEF6E002-A874-101A-88BA-00AA00200CAB
```

Natürlich kann der Entwickler, der für sich diese ID generieren lässt, nicht hundert prozentig sicher sein, dass noch niemand sonst diese ID verwendet hat. Allerdings ist es bei einem Wertebereich der alle Zahlen von 0 bis 2^{128} umfasst sehr unwahrscheinlich, wenn auch nicht ausgeschlossen. Um diese Möglichkeit noch weiter zu reduzieren, bietet die Windows API die Funktion ”CoCreateGuid” zur Erzeugung der ID. Ausserdem erleichtert dies natürlich dem Entwickler das Handling.

Die ID verwendet man, um die Klassen, Anwendungen und Typebibliotheken eindeutig zu identifizieren auch über Rechnergrenzen hinaus. Deshalb erfordert auch jede neue Version einer Typebibliothek zwingend eine neue ID erhalten.

In Windows wird die Zuordnung der ID zu den in der IDL, beziehungsweise Typebibliothek beschriebenen Definitionen in der Registratur (engl. registry) gespeichert, wenn sie im System registriert wird. Um dem Anwender den Zugriff zu erleichtern, gibt es zusätzlich die ProgID, deren Zuordnung zur GUID ebenfalls in der Windows Registry gespeichert ist. Diese ermöglicht zum Beispiel den Zugriff auf MS Word Version 8 mit der ProgID ”Word.Application.8” ohne die CLSID zu kennen.

2.1.3 Verarbeitung von Ereignissen

Während bei der klassischen Client-Server-Kommunikation (Abbildung 2.2) der Informationsfluss eher einseitig von statten geht, da der Server im Normalfall die Daten liefert die der Client entweder ereignisgesteuert oder zyklisch abruft, so besteht mit COM auch die Möglichkeit der Benachrichtigung durch Ereignisse (engl. events). Mit diesem Mechanismus (Abbildung 2.3) ist der Server in der Lage, den Client aktiv über eine Veränderung eines Zustandes oder Wertes zu informieren, also selbst eine Aktion zu initiieren.

Der Client muss natürlich so gestaltet werden, dass er in der Lage ist, diese Ereignisse

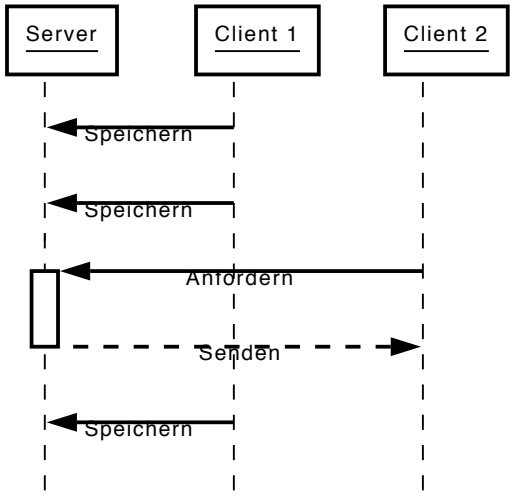

Abbildung 2.2: Kommunikation ohne Events

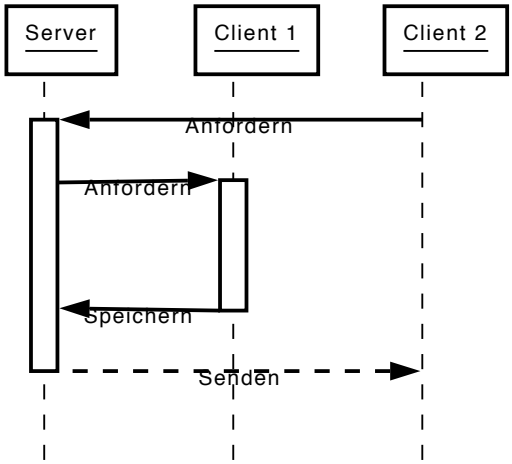

Abbildung 2.3: Kommunikation mit Events

zu registrieren. Dazu werden entsprechende Methoden implementiert mit denen der Client dem Server anzeigt, welche Nachrichten er bereit ist zu empfangen. Hat er sich damit einmal beim Server gemeldet kann dieser nun seinerseits aktiv werden und gegebenenfalls eine Verbindung zur Benachrichtigung herstellen.

Dies ist auch in n:m Konstellation möglich. Es ist also mehreren Clients gestattet, ihre Empfangsbereichtschaft zu signalisieren. Damit wird dem Server erlaubt, sie alle über eine entsprechende Nachricht zu informieren. Dabei ist diese Standard Kommunikation über das Vorhandensein der Schnittstellen sehr allgemein gehalten, um es unabhängig von den übermittelten Daten zu gestalten. Es stellt nur eine logische Verbindung zwischen den beiden Objekten her, völlig irrelevant wie deren Schnittstellen konkret aussehen. Außerdem bietet diese Art der Anbindung die Möglichkeit zum Variieren während der Laufzeit. Dies bedeutet, dass der Client während der Laufzeit dem Server mitteilen kann, dass er zum Beispiel über dieses Event nicht mehr auf dem laufenden gehalten werden möchte, aber stattdessen ein anderes Ereignis sozusagen abonniert.

2.1.4 Rollenverteilung bei der Kommunikation

Bei der Nutzung der von COM vergegebenen Mechanismen ergeben sich grundsätzlich zwei Anwendungsfälle:

- Server

- Client

Der Client zeichnet sich im Besonderen dadurch aus, dass er eigentlich nur die angebotenen Dienste des Servers nutzt, also Anfragen sendet und die Ergebnisse abfragt. Die Ergebnisse können einfache Datentypen oder wiederum Referenzen auf eine andere COM-Klasse sein.

Der Server bietet diese Dienste an. Daher müssen seine Schnittstellen, wie in den vorigen Abschnitten beschrieben, spezifiziert werden und die Schnittstellendefinition entsprechend abgelegt sein. Vor allem ist es wichtig, dass im System die Typelib registriert ist, damit eine Verbindung zustande kommen kann. Außerdem ist zu erkennen, dass es verschiedene Arten von COM-Servern gibt, die sich im Grunde anhand ihrer Instanziierung unterscheiden. So existieren Server die mehrmals gestartet werden können, also von denen mehr als eine Instanz zur gleichen Zeit existieren kann (Abbildung 2.4) oder Server von denen immer nur genau eine Instanz existiert (Abbildung 2.5). Für erstere Server (Inproc-Server) wird bei jedem Verbindungsaufbau (engl. connect) eines Clienten auf die angebotene COM-Schnittstelle des Server ein neues Programm gestartet, das im gleichen Context wie der aufrufende Prozess läuft.

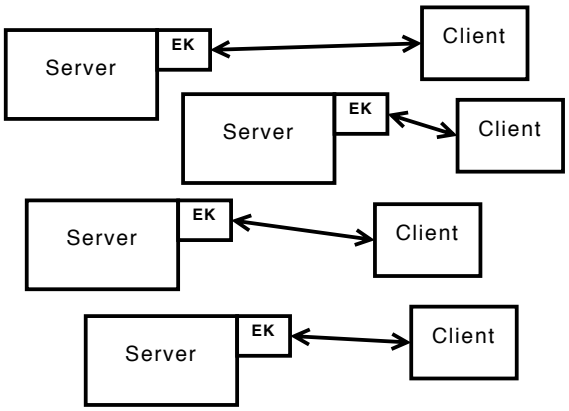

Abbildung 2.4: Inproc-Server

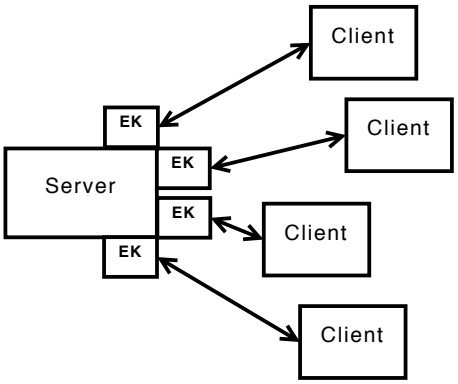

Abbildung 2.5: lokaler Server

Bei Servern der Kategorie zwei (lokaler Server) wird nur der Zugriff des ersten Clienten dazu führen, dass eine Instanz des Programmes erzeugt wird. Jeder weitere Verbindungsversuch erzeugt zwar eine neue Instanz der Entry-Klasse, diese Klasse ist aber Mitglied der einmaligen Instanz der angesprochenen Anwendung. Normalerweise nutzt man diese Klasse als Factory zum Erzeugen der Referenzen zu bestehenden Daten der Programmreferenz. Zum Beispiel ist es möglich, auf die programmeigenen Daten über eine globale Variable/Klasse zuzugreifen oder über eine Klasse die als Singleton implementiert ist.

Die meisten Programme mit einer GUI[1], die den Zugriff auf ausgewählte Funktionen über COM ermöglichen, können nur einmalig erstellt werden und sind damit also von der zweiten Kategorie. Natürlich erhält man beim Nutzen der Automatisierungsschnittstelle von CARMEN-TopEd auch nur Elemente einer einzigen Instanz des Programmes.

Beim Nutzen komplexerer Programmier- beziehungsweise Scriptsprachen erleichtert sich teilweise der Prozess der Schnittstellendefinition(IDL). So wird in Visual-Basic zum Beispiel die Typelib automatisch generiert und in das binäre Compilat integriert, um mit den typischen Windows-Tools („regsvr23.exe“) im System registriert zu werden. Dies hat zwar den Vorteil, dass der Entwickler diesem Arbeitsschritt überspringen kann, allerdings entzieht es ihm auch ein Stück Kontrolle, da alle Klassen die öffentlich („public“) deklariert sind über COM auslesbar werden.

2.2 Programmiersprache Python

Python ist eine plattformunabhängige und objektorientierte Programmiersprache, die im Bereich der Softwareentwicklung ein breites Anwendungsspektrum findet. Dies reicht von klassischen Kommandozeilen-Programmen bis hin zu Anwendungen für Datenbanken, Internet und Netzwerk sowie grafischen Benutzeroberflächen für Windows, Linux, MAC und anderer Systeme. Python unterliegt einer Open-Source Lizenz, so dass es für privaten und kommerziellen Gebrauch frei nutzbar ist [Pyt].

```
def funcName(self, attribut):
        if (1 == 1):
                print "1==1"
```

Abbildung 2.6: Beispiel eines Funktionsaufruf in Python mit if-then Block

Den grössten Unterschied, den man bei Python auf den ersten Blick erkennen kann, ist der Verzicht von Klammern zur Strukturierung von Blöcken (Abbildung 2.6). Diese wird in Python ausschließlich durch Einrückung bewerkstelligt. Dadurch kommt es zu einer sehr sauberen Strukturierung des Quelltextes, die auch für den Anfänger sehr gut nachzuvollziehen ist.

Als weitere Besonderheit sind die dynamischen Datentypen zu nennen, denn die Typüberprüfung wird erst zur Laufzeit vorgenommen. Dabei unterstützt Python primitive

1 emgl. graphical user interface - grafische Benutzerschnittstelle

Datentypen, wie Integer, Float, Bollean und String genauso, wie komplexe Strukturen, wie Listen oder Dictionaries. Dictionaries sind im Grunde genommen Hashmaps, deren Elemente ein Tupel aus Schlüssel und Wert ist. Der Wert kann wiederum ein beliebiger Datentyp sein, auch ein komplexer Typ ist möglich.

Natürlich ist Python eine objektorientierte Sprache in der auch Mehrfachvererbung und komplexe Klassenmodelle verwendet werden können. Aber für einfache Scripte ist es ebenso möglich, eine rein sequentielle Implementation zu wählen. Außerdem unterstützt Python ebenso Ausnahmen (Exceptions) und Introspektion (beziehungsweise Reflexion). Dies bedeutet, dass das Programm selbst Kenntnisse über seine eigene Struktur erlangen kann.

In der Standard-Python Distribution ist bereits ein Vielzahl von Paketen enthalten, aber zusätzlich sind im Internet unzählige Module verfügbar, die installiert werden können. Ist für einen Anwendungsfall eine Bibliothek nötig, so kann diese mittels der Python-API erzeugt und danach eingebunden werden. Diese Erweiterungen werden üblicherweise in C beziehungsweise C++ programmiert. Für nähere Details zu Python sei hier auf weiterführende Literatur verwiesen [Wei05], [MN05], [Lut99].

2.2.1 wxPython

Das wxPython Toolkit erleichtert die Erzeugung einer grafischen Benutzeroberfläche, das so genannte *Graphical User Interface*(kurz GUI). Dies basiert auf der *wxWidgets* Bibliothek die in C++ geschrieben ist und für die meisten Betriebssysteme (Windows, Linux, MacOSX ...) zur Verfügung steht.

Durch die wxPython API ist es möglich, ohne grossen Aufwand moderne und leistungsfähige Bedienoberflächen zu erzeugen. Dabei erhält man genauso Zugriff auf Standarddialoge (wxFileDialog, wxColourDialog ...), als auch auf Elemente, wie Listen (wxListCtrl), Bäume (wxTreeCtrl), Textfelder und Schaltflächen.

2.2.2 Python und COM

Um in Python den Zugriff auf die Betriebssystem-Funktionen und Schnittstellen zu erlangen, gibt es die win32 Erweiterung. Speziell win32com ermöglicht das Benutzen der COM-Schnittstelle, wobei beide Rollen, Server und Client, möglich sind.

Besonders hervorzuheben ist hierbei die Introspektions Eigenschaft und die dynamischen Datentypen von Python, die das Nutzen von COM sehr vereinfacht. Dadurch sind übertragen Werte direkt zu benutzen, ohne die Spezifikation aufwendig über die IDL Datei einzulesen. Der Python Interpreter erzeugt selbständig aus der registrierten Typbibliothek der angebotenen Schnittstelle eine entsprechende Quellcoddatei. Wenn komplexe Klassen zur Kommunikation benötigt werden, kann diese importiert werden und man hat Zugriff auf die realisierte Struktur.

```
class comSever:
    _public_methods_ = ['Methode1','Methode2']

    _reg_progid_ = "myComserver.Application"
    _reg_clsctx_ = pythoncom.CLSCTX_LOCAL_SERVER
    _reg_desc_ = "ein Beispiel COM Server"
    _reg_clsid_ = "{F9ABDC2F-9D95-4C9C-B120-8F65A0AE73A9}"

def main():
    print "Registering COM server ... "
    import win32com.server.register
    win32com.server.register.UseCommandLine(comSever)

if __name__ == '__main__':
    main()
```

Abbildung 2.7: Beispiel eines COM-Servers in Python

Auch das Erstellen eines COM-Servers gestaltet sich im Vergleich zu C++ sehr viel unkomplizierter. Zusätzlich zum Importieren des Paketes muss eine Klasse zum Einstieg deklariert werden. Dabei wird mit mehreren Attributen die Art des Servers (Inproc et cetera), der Name, die IDL und eine Beschreibung festgelegt. Die zur Verfügung gestellten Methoden beziehungsweise Attribute werden durch einfache Listen veröffentlicht. Sobald der Server registriert wurde, kann er auch schon verwendet werden. Eine Beispiel Implementierung ist in Abbildung 2.7 dargestellt.

3 Entwurf der Schnittstelle

3.1 Realisierungsvarianten

Bei der Umsetzung des Projektes waren im Vorfeld mehrere verschiedene Lösungsvarianten offensichtlich, deren Realisierbarkeit durch die technischen Möglichkeiten der zu verbindenden Software dann jedoch eingeschränkt wurde. Dazu mussten vor allem zwei grundlegende Dinge vorgesehen werden:

1. die Schnittstelle zwischen ECU-TEST und CARMEN, um CARMEN Testablauf zu steuern und Daten zwischen den Modulen auszutauschen

2. das CARMEN- Modul, welches die gewünschte Funktionalität zur Busüberwachung in der CARMEN Umgebung zur Verfügung stellt. (Checksummprüfung, Alivecounterüberwachung, et cetera)

Um die Steuerung in ECU-TEST zu ermöglichen, hat das Modul adäquate Schnittstellen anzubieten, mit denen es möglich ist, die internen Daten von außen so zu verändern, dass damit der Testablauf in CARMEN-TopEd auch wirklich beeinflusst wird. Der eigentliche funktionelle Umfang des zu testenden Zusammenhangs, wie zum Beispiel ein Alive-Check oder eine CRC-Prüfung, kann im CARMEN-Modul implementiert werden. Als ECU-TEST Testschritt ist dafür ein Setzen der Parameter und danach ein Auslesen der Ergebnisse vorgesehen.

Als Kommunikationsmethode zwischen beiden Softwarepaketen bietet sich die Windows COM- Schnittstelle[1] an, da diese von CARMEN zur Automatisierung angeboten wird. Im Grunde gibt es zwei mögliche Wege des Datenaustausches zwischen CARMEN-Modulen und ECU-TEST:

1 Component Object Model Technologies

1. Die Kommunikation der Module mit ECU erfolgt indirekt über einen separaten COM-Server und die Automatisierungsschnittstelle von CARMEN wird nur zur Steuerung von CARMEN selbst verwendet.

2. Die Module der CARMEN Konfiguration werden direkt über die Automatisierungsschnittstelle von CARMEN angesprochen, da diese auch für das Starten/Stoppen des Tests verwendet wird.

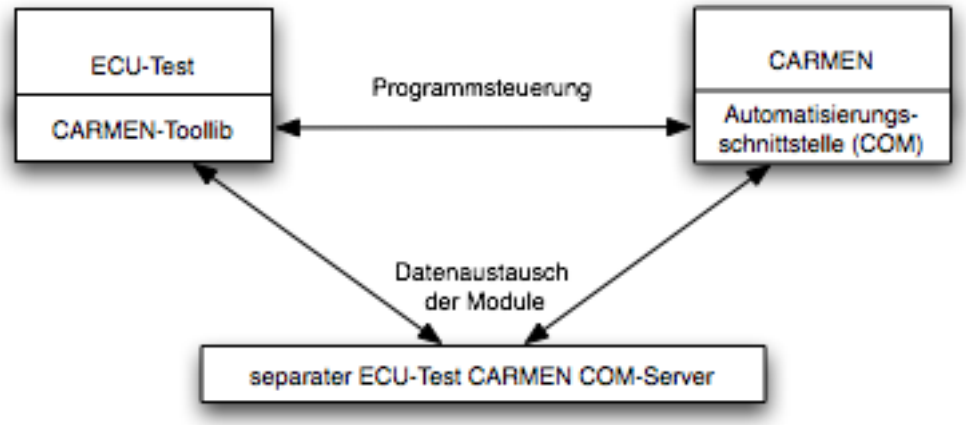

Abbildung 3.1: indirekter Datenaustausch mit separaten COM-Server

Bei der Variante 1 (Abbildung 3.1) liegt der Vorteil bei der Flexibilität der Implementierung, da selbst bestimmt werden kann, in welcher Art und Weise auf die Daten des jeweiligen Moduls zugegriffen werden soll. Das Modul agiert aktiv und setzt die Daten zu selbst bestimmten Zeitpunkten. Zusätzlich wäre auch eine Benachrichtigung der auslesenden Anwendung, zum Beispiel durch Events möglich. Allerdings muss dafür auch ein zusätzlicher COM- Server integriert werden, auf den beide Programme - ECU-TEST und das CARMEN Modul zugreifen können. Dieses kann aber auch zusätzliche Probleme durch Synchronisation und allgemeine Fehleranfälligkeit bereiten. Da jeweils beide Programmpakete mit dem zusätzlichen Server kommunizieren sollen, muss dieser installiert werden und vor dem Zugriff ist es erforderlich eine Prüfung auf das Vorhandensein des Servers durchzuführen werden.

Ein ebenso kritisches Problem dieser Realsierungsvariante ist die Datenkonsistenz. Da der Server darauf angewiesen ist, immer die aktuellsten Daten zugewiesen zu bekommen, kann er diese nicht aktiv selbst anfordern. Sollte nun zum Beispiel das Modul eine unvorhergesehene Fehlfunktion aufweisen und sich nicht ordnungsgemäß abmelden, kann der Server nur veraltete Informationen weitergeben. Dieses Problem könnte mit

Hilfe von Events verringert werden.

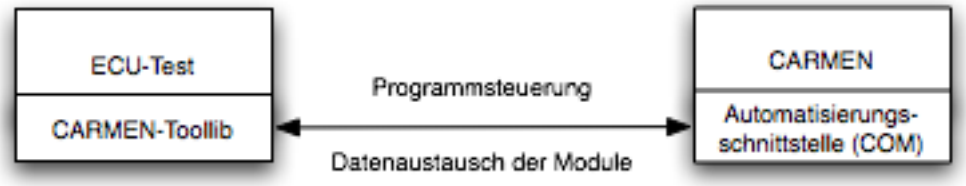

Abbildung 3.2: direkter Datenaustausch mit CARMEN

Der Vorteil der Lösung 2, dargestellt in Abbildung 3.2, ist offensichtlich die Einfachheit der Kommunikation, da direkt über den gleichen Weg der Datenaustausch und die Steuerung von CARMEN stattfindet. Es müssten auch im Modul nur wenige Attribute und Methoden implementiert werden, um für den Austausch der Parameter und Rückgabewerte zu sorgen.

Nachteilig ist natürlich, dass die Grenzen der Kommunikation durch die Schnittstellen von CARMEN definiert sind, auf die nur die CARMEN Entwickler direkten Einfluss haben. Dafür würden aber auch die mitgelieferten CARMEN- Module unterstützt und nicht auf die speziellen Module eingeschränkt, die für die ECU-TEST Kommunikation vorbereitet sind. Auch das Problem der Synchronisation ergibt sich nur bedingt, da ECU-TEST nur im Polling Verfahren die Informationen von dem Modul liest. Allerdings sind die gelesenen Informationen durch den direkten Zugriff auf Modul- Attribute ausschließlich zum Lese-Zeitpunkt aktuell.

Da zum Beginn der Studienarbeit nur CARMEN in der Version 1.1 zur Verfügung stand und diese den vorgesehenen direkten Zugriff auf die Instanzen der Module nicht ermöglichte, wurden beide Varianten eingearbeitet. Allerdings ist die Lösung 2 nur mit der aktuellen Version 1.2 von CARMEN nutzbar. Da durch die oben genannten Vor- und Nachteile die direkte Lösung bevorzugt wird, wurde von der indirekten Variante nur eine minimale Version eingebaut. Diese unterstützt selbstverständlich den Austausch von Parametern und Rückgabewerten, allerdings ist die Synchronisation durch Events und erweiterter Austausch von Objekt- Instanzen als Ergebnis von Methoden-Aufrufen nicht realisiert.

3.2 ECU-TEST Integration

Damit die zur Integration in ECU-TEST nötigen Modifikationen deutlicher werden, soll in den folgenden Abschnittem näher auf den Aufbau und die Kernelemete des Testautomatisierungswerkzeugs ECU-TEST eingegangen werden. Dabei wird auch beschrieben, welche Veränderungen zur Erfüllung der Aufgabenstellung nötig werden.

3.2.1 Aufbau des Testautomatisierungstools

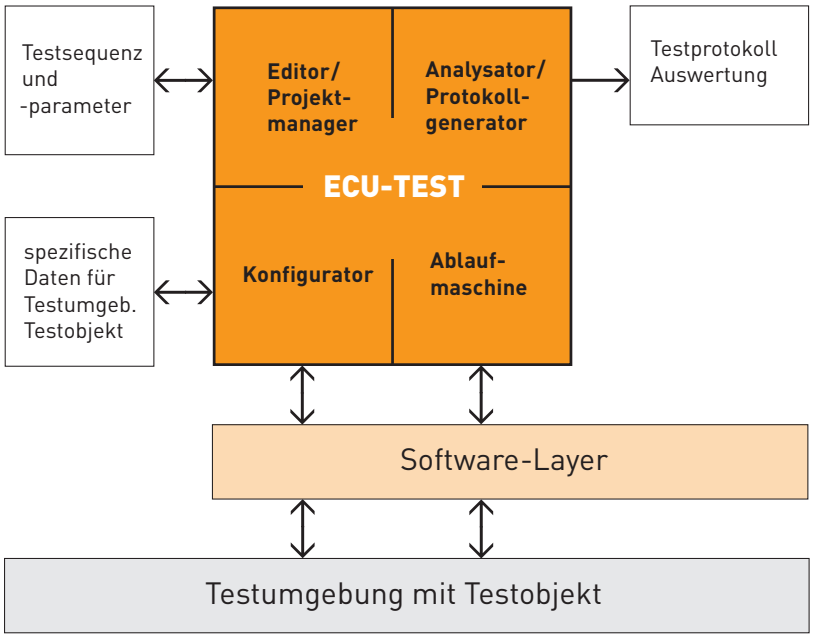

Abbildung 3.3: ECU-TEST - Architektur

Aus Sicht des Testingenieurs ist der Editor zum Erstellen der Testpakete der Hauptbestandteil von ECU-TEST. Aber selbstverständlich gehören auch die Elemente zur Konfiguration der gewünschten Parameter und Anbindung externer Software Utilitys, sowie die Generierung der Dokumentation zur Testauswertung und ebenso die eigentliche Engine zur Ausführung der Testschritte zum Kern von ECU-TEST.

Editor Im Editor werden Test-Packages verwaltet, die Testschritte zugefügt und organisiert. Dazu existiert ein Hauptfenster, in welchem die Testschritte des aktuell geöffneten Packages dargestellt werden (Abbildung 3.4 - Bereich 2). Auf der linken Seite ist ein

Reiter (Abbildung 3.4 - Bereich 1), in dem alle verfügbaren Testschritte aufgelistet werden. Diese sind logischerweise durch Tabulatoren je nach Thema getrennt, damit die Übersicht für den Testingenieur erhalten bleibt.

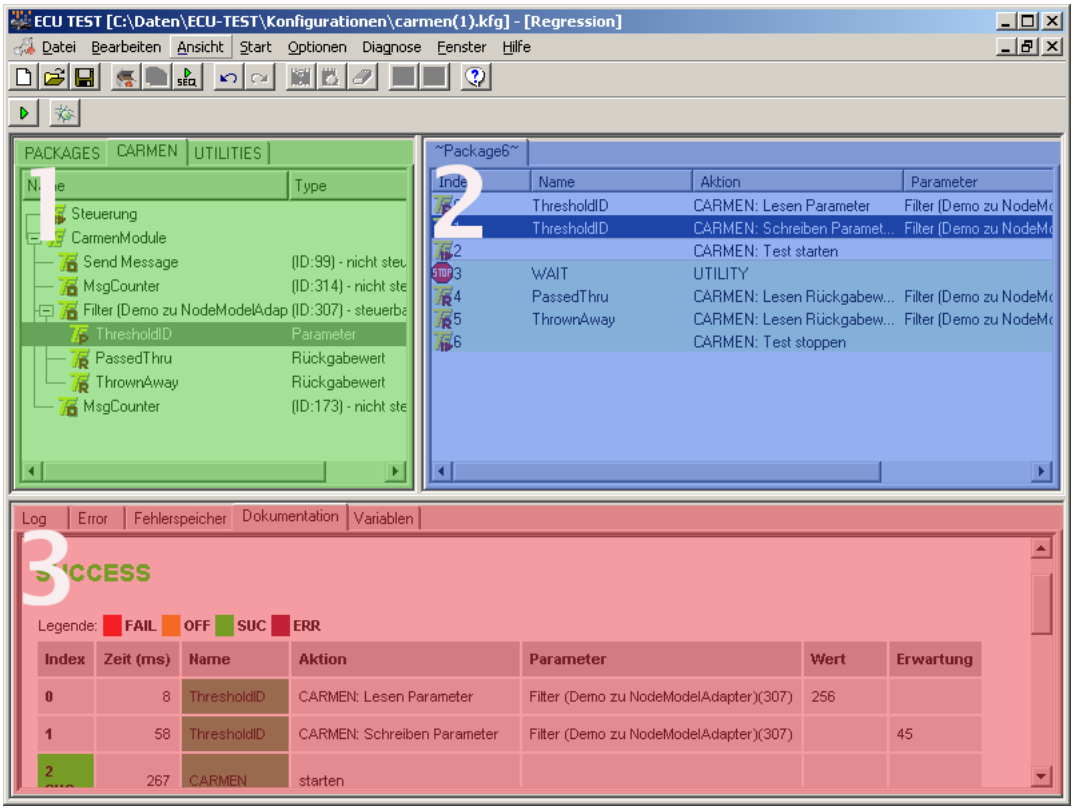

Abbildung 3.4: ECU-TEST - Regression

Von diesem Reiter könnnen die Testschritte per Rechtsklick in das aktuelle Test-Package übernommen werden. Beim Einfügen öffnet sich ein Dialog, in dem die verbleibenden Parameter des Testschrittes konfiguriert werden. Die Parameter können selbstverständlich auch später erneut verändert werden, indem im Testpackage der Schritt doppelgeklickt wird. Das Fenster im unteren Bereich (Abbildung 3.4 - Bereich 3) stellt je nach Situation verschiedene Dinge dar, wie den Testreport oder die ECU-TEST Logdatei.

Analysator/Protokollgenerator Der Testreport wird als XML[1] oder HTML[2] Datei generiert. Damit eine bessere Lesbarkeit des XML Files gegeben ist, kann diese durch verschiedene Stylesheet Daten in HTML oder PDF umgewandelt werden. Au“serdem ist eine interne HTML Generierung in ECU-TEST integriert, allerdings müssen dort für

1 extensible markup language
2 hypertext markup language

jede unterschiedliche Art von Testschritten entsprechende Anweisungen zum Festlegen der Formatierung implementiert werden. Welche Art der Dokumentation erzeugt werden soll, kann in der Konfiguration festgelegt werden.

Konfiguration Vorgaben wie die Pfade zum Speichern der Packages, verwendete Busse und dafür ötige Busdatenbanken, angesprochene Softwaremodelle et cetera werden in der Konfiguration eingestellt. Dafür können auch verschiedene Konfigurationen für unterschiedliche Anwendungen gespeichert werden, deren Einstellungen beim Öffnen der Regresseion übergeben werden. Die unterschiedlichen Themenbereiche sind in der Konfiguration durch verschiedene Reiter unterteilt.

Zur Parametrierung des Zugriffs auf externe Softwarewerkzeuge ist der Bereich Software der Konfiguration vorgesehen. Dort sind die tooleigenen Einstellungen vorzunehmen und die Art, wie die Verbindung zur Toollib hergestellt wird. Dabei kann man entscheiden, lokal zuzugreifen oder über den Applikationsserver, wofür dann die Addresse des Toolhosts anzugeben ist.

Ablaufmaschine Die Testschritte werden sequenziell in der angegebenen Reihenfolge ausgeführt. Trotzdem sind komplexere Strukturen durch Schleifen und Entscheidungsalternativen möglich. Durch die Möglichkeit einzelne Packages als Teil eines anderen Packages einzufügen, können komplexe Testaufgaben in kleinere Problemlösungen segmentiert werden.

3.2.2 Nötige Modifikationen

Toollib Die Routinen zur Kommunikation mit den externen Programmen über COM oder andere Schnittstellen, sind in der so gennanten Toollib realisiert. Dort findet die Verbindung zu der Anwendung und die Übermittlung der Daten statt. Damit die für die Toollib nötigen Parameter korrekt konfiguriert werden können, müssen außerdem entsprechende Dialoge für die Softwarekonfiguration implementiert sein.

Um die Toollib auch über den ECU-TEST eigenen Applikationsserver anzusprechen, ist es nötig für die Methoden der Toollib zusätzliche Wrapper Klassen zu erstellen. Über diese Wrapper Klassen findet die Verbindung zur Regression statt, die dazu implementiert werden muss. Durch den Applikationsserver ist es möglich, Anwendungen auch auf entfernten Rechnern zu nutzen, die Verbindung wird über das Netzwerk hergestellt.

Für die in Abschnitt 3.1 erläuterten Lösungsalternativen hätten unterschiedliche Toollibs implementiert werden können. Es wurde aber vorgezogen, beide Varianten in einer einzigen Datei zu integrieren. Nach der Parametrierung in der Softwarekonfiguration wird die Kommunikation zwar unterschiedlich gestaltet, die Schnittstellen der Toollib sind aber identisch für beide Lösungsvarianten.

Regression Damit der Nutzer die Testschritte in sein Package einfügen kann, ist es erforderlich eine passende Liste, die die gewünschten Schritte anzeigt zu erstellen. Teilweise können vorhandene Listen wie der Bus Viewer verwendet werden. Für dieses Projekt bot es sich an, eine komplett neue Ansicht zu integrieren, um alle Fuktionen für den Nutzer anzubieten. Um von dort Schritte einzufügen, muss für jeden einzelnen Testschritt ein Dialog zum Editieren der Parameter realisiert werden. Es kann teilweise auf Bauteile wie den Editor der Vergleichswerte zurückgegriffen werden, aber zum Grossteil müssen die gewünschten Eingabefelder selbst gestaltet werden.

Damit die Daten der Testschritte verarbeitet werden können, ist es nötig für alle Testschritte entsprechende Datencontainer zu erstellen, in denen die Parameter des Schrittes und die Ergebnisse bei der Testdurchführung abgelegt werden. Über eine einzelne Testschritt-Klasse werden von der Regression aus diese Datencontainer verwaltet. Durch die Wrapper Klassen wird die Verbindung zur Toollib genutzt, um die Testdurchführung zu realisieren und die Testschrittergebnisse zu ermitteln. Wenn diese ordnungsgemäß in den Container abgelegt sind, werden sie direkt in der Regression mit dargestellt und bei der Erzeugung des Reports ausführlich dokumentiert.

Damit die Reportdarstellung korrekt formatiert ist, sollte ebenfalls an dieser Stelle die nötigen Ergänzungen vorgenommen werden.

3.3 Entwurf eines Moduls zur Busüberwachung

Der gravierende Unterschied zwischen beiden Werkzeugen, ECU-TEST und CARMEN, ist ihre unterschiedliche Testphilosophie. CARMEN arbeitet eher ereignissorientiert wenn man die einzelnen Busnachrichten als Ereignisse versteht. Der Nutzer gestaltet die CARMEN-Konfiguration so, dass von einem oder mehreren Interfaces echte oder simulierte Bussignale über Leitungen verbunden, durch verschiedene Module geleitet werden. Dabei gibt es unterschiedliche Module, die teilweise den Datenstrom verändern können wie zum Beispiel Filter und andere, die ihn nur beobachten, wie zum Beispiel ein einfacher Message Zähler.

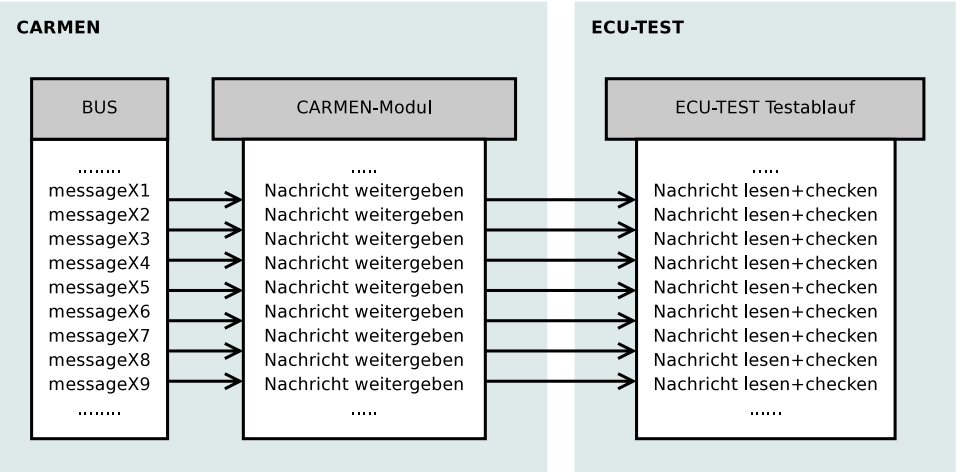

Abbildung 3.5: Nachrichtenüberprüfung in ECU-TEST

ECU-TEST dagegen arbeitet sequentiell. Um Nachrichten zu überprüfen gibt es im Grunde zwei Möglichkeiten. Entweder die Nachrichten werden aus CARMEN nach ECU-TEST gesendet und dort überprüft (Abbildung 3.5) oder der Test auf spezifische BUS oder Nachrichten Eigenschaften werden in CARMEN vorgenommen und nur die Ergebnisse zu ECU-TEST übermittelt (Abbildung 3.6). Mit der entworfenen Schnittstelle sind theoretisch beide Varianten möglich. Allerdings ist nur die zweite Variante für eine Überprüfung vieler Nachrichten gleichzeitig geeignet. Da die Verbindung über COM verhältnismäßig langsam ist, würde sie sich bei der Variante eins, also dem Transfer der Nachrichten, als Flaschenhals erweisen.

Da mit einem entsprechenden CARMEN-Modul eine Vielzahl von Nachrichten Eigenschaften, wie CRC, Alive-Zähler, Zeitstempel et cetera geprüft werden kann, wäre es unsinnig alle am Modul eintreffenden Nachrichten grundsätzlich zu testen. Außerdem werden zum Test spezifischer Eigenschaften zusätzliche Parameter benötigt. Daher muss für eine sinvolle Testauswahl eine Registrierung der interessanten Nachrichten mit ihren Paramtern stattfinden. Dann kann nach einer angemessenen Wartezeit, in der der Test in CARMEN abläuft und die konfigurierten Überprüfungen für die ausgewählten Busnachrichten in CARMEN stattfinden, die Ergebnisse von ECU-Test ausgelesen werden.

Mit diesem Vorgehen ist es auch möglich alle Busnachrichten in Echtzeit zu überprüfen, wenn die Busse mit der üblichen Geschwindigkeit arbeiten. Notwendigerweise ist hierfür das Modul selbst in C++ zu implementieren. Zum Testen der Schittstelle und Darstellung der nötigen Modifikationen wurde ein Modul in Visual Basic realisiert, allerdings ist das

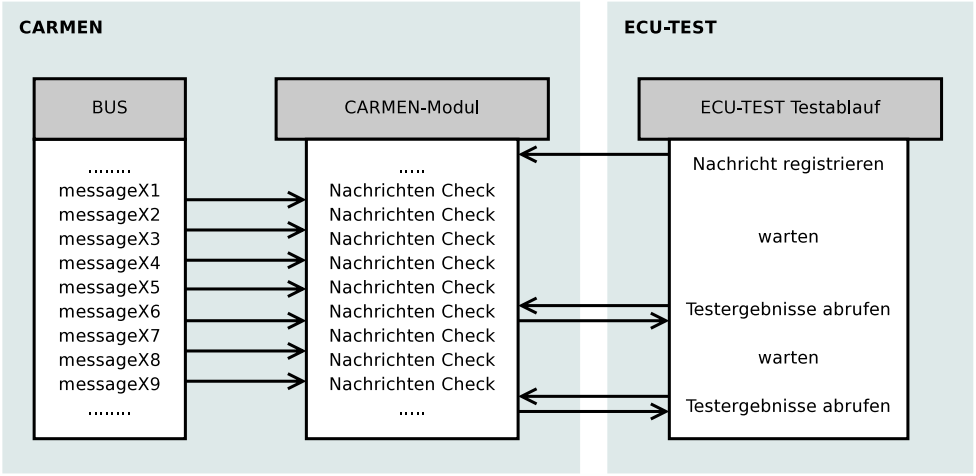

Abbildung 3.6: Nachrichtenüberprüfung in CARMEN

nicht für eine vollständige Nachrichten Überprüfung geeignet, da bei dieser Variante das Modul selbst die einzelnen Nachrichten von CARMEN per COM erhält und daher nicht performant genug arbeitet.

Ein solches Modul benötigt also Parameter oder Methoden, mit denen die Nachrichten registriert werden können und Rückgabewerte oder Methoden, um die Testergebnisse auszulesen. Schon an dieser Stelle zeigt sich der Vorteil der direkten Schnittstelle (Variante 2), da auch Methoden genutzt werden können, die den Vorteil des zielgerichteten Auslesens der Ergebnisse ermöglichen (also Testergebnisse einer einzelnen Nachricht) und durch den Aufruf der Methoden mit mehreren Parametern der Registrierungsvorgang sehr viel klarer definiert ist. Eine andere Möglichkeit ist das Setzen von Paramtern. Das erfordert aber das Erstellen komplexer Strings welche die Eigenschaften beschreiben.

Bei der direkten Anbindung ist ebenso die Rückgabe komplexer Datentypen, wie zum Beispiel kompletter Klassen möglich. Das erleichtert den Umgang wesentlich. Im Ergebniss der Analyse beider Arten, der direkten und indirekten Anbindung, ist es möglich ein entsprechendes Modul zu realisieren.

4 praktische Umsetzung der Anbindung

Die endgültige Implementierung der Lösungsvariante 2 besteht aus drei Teilen:

1. ECU-TEST-CARMEN-COM-Server zum Datenaustausch beider Programmpakete
2. die Test-Klassen und Toollib für ECU-TEST
3. CARMEN-Modul

Für die direkte Anbindung der Module sind fast identische Elemente nötig, nur auf den COM- Server kann verzichtet werden, da CARMEN selbst für die Kommunikation zur Verfügung steht.

Die CARMEN Module kann man wahlweise in C++ oder Visual Basic programmieren, letzteres bietet sich der Einfachheit halber zum implementieren des Test-Modules an. Da ECU-TEST in Python programmiert ist und diese Script- Sprache dank ihrer Objektorientierung leicht zu Handhaben und trotzdem sehr übersichtlich ist, wird auch der COM-Server in Python implementiert.

4.1 ECU-TEST-CARMEN-COM-Server

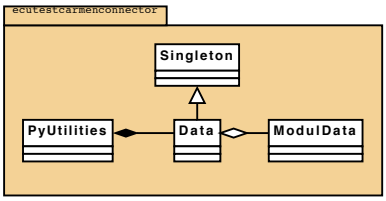

Abbildung 4.1: UML-Diagramm des ECU-TEST-CARMEN-COM-Server (vereinfacht, vollständig in Abbildung A.1)

Der COM-Server besteht im Grunde aus drei Klassen (Abbildung4.1). Die Klasse ”PyUtilities”, ”ModulData” und ”Data”. Diese sind für die Datenhaltung und den Austausch der selbigen verantwortlich.

4.1.1 Klasse ”PyUtilities”

Die Klasse ”PyUtilities” ist die Hauptklasse der Anwendung. Sie ist für die Freigabe der Methoden des COM-Servers nach außen verantwortlich. Daher wird sie auch als ”ECU-TestCarmenCOM.Application” registriert. Die eingearbeiteten Methoden spiegeln sich fast identisch in den anderen beiden Klassen wieder. Außerdem speichert diese Klasse die aktuelle ID des zuletzt registrierten Moduls. Damit ist es sehr viel unkomplizierter im CARMEN Modul auf die entsprechenden Modul-Daten zuzugreifen, da jedes Modul eine eigene Instanz des COM- Servers zugewiesen bekommt.

4.1.2 Klasse ”Data”

”Data” ist als ”Singleton” Klasse implementiert. Dadurch ist sichergestellt, dass jede Instanz des COM- Servers auf die gleiche Liste von Daten zugreift, da alle Instanzen nur eine gemeinsame Instanz von Modul Data zur Verfügung haben. Die wichtigste Methode dieser Klasse ist ”getModules”, sie liefert eine Liste mit den Namen der momentan registrierten Module zurück. Diese nutzt ECU-TEST um bei der Anzeige zu erfahren, welche Module verfügbar sind und danach Informationen über die jeweils möglichen Rückgabewerte (”getReturnNames”) und Parameter (”getParamNames”) zu lesen.

Die genannten Funktionen liefern eine Liste mit den ausgewählten Namen der Rückgabewerte oder Paramter. Mit den Methoden ”getReturnValues” und ”getParamValues” erhält man Arrays mit den korrespondierenden Werten die sich jeweils an der gleichen Postion befinden wie der entsprechende Name. Natürlich sind auch Methoden implementiert, mit denen man auf einzelne Werte direkt zugreifen kann (”getReturnValue” und ”getParamValue”) und selbstverständlich sind genauso die äquivalenten Set-Methoden vorhanden.

Wie schon erwähnt muss sich ein CARMEN- Modul bei seiner Implementierung (zum Beispiel beim Einfügen in die CARMEN- Konfiguration oder beim Öffnen der Konfiguration) beim COM- Server registrieren. Dabei wird dem Data-Container ein neues ”Modul-Data” Objekt zugefügt und die entsprechende ID zurückgeliefert.

4.1.3 Klasse "ModulData"

Diese Klasse ist ein Container, welcher alle Daten der einzelnen Module speichert. Dabei werden jeweils für Rückgabewerte, und Paramter eine Liste der Namen und Werte vorgehalten. Somit ist es möglich die jeweiligen Tupel auszulesen oder zu überschreiben. Die Liste der Namen wird im Normalfall nur einmalig gefüllt, da sich die Bezeichnungen der Paramter beziehungsweise Rückgabewerte während der Lebensdauer der Modul-Instanz nicht mehr ändert. Die Werte dagegen werden selbstverständlich regelmäßig oder ereignisorientiert immer wieder überschrieben.

Als Datencontainer besitzt diese Klasse natürlich die Set- und Get-Methoden, um die Elemente der Liste Auszulesen oder zu Veränderen, damit ein transparenter Zugang ermöglicht wird. Es ist alßo kein direkter Zugang auf die Arrays implementiert.

4.2 Test-Klassen und Toollib für ECU-TEST

Wie in Abschnitt 3.2 beschrieben ist, werden durch die Komplexität von ECU-TEST zur Integration mehrere Dateien benötigt. Diese sind für die Konfiguration und Durchführung der Testschritte verantwortlich. Zur Nutzung des Toollib über den ECU-TEST Applikationsserver beziehungsweise Windows COM-Schnittstelle müssen die übertragenen Klassen außerdem entsprechend gewrapt werden.

1. die Toollib und das zugehörige Panel zur Konfiguration:
   - ToolLibs/CARMENlib.py
   - Dialogs/Configuration/SoftwareTools/CARMENlibPanel.py
   - Dialogs/Configuration/SoftwareTools/EntryCarmen.py
2. die Testklasse und CARMEN-Client:
   - TestClassCarmen.py
   - CARMENclient.py
   - CARMENViewer.py

3. die Teststep-Klassen und zugehörigen Konfigurationsdialoge:

   - _app/carmen/TsControlCarmen.py
   - _app/carmen/DialogConfTsControlCarmen.py
   - _app/carmen/TsReadReturn.py
   - _app/carmen/DialogConfTsReadReturn.py
   - _app/carmen/TsReadParam.py
   - _app/carmen/DialogConfTsReadParam.py
   - _app/carmen/TsWriteParam.py
   - _app/carmen/DialogConfTsWriteParam.py
   - _app/carmen/TsFuncCall.py
   - _app/carmen/DialogConfTsFuncCall.py

4.2.1 Toollib und das zugehörige Panel zur Konfiguration

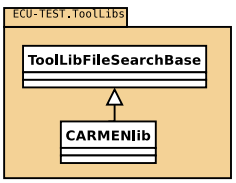

Abbildung 4.2: UML-Diagramm der Toollib (vereinfacht, vollständig in Abbildung A.2)

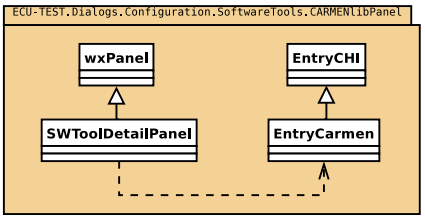

Abbildung 4.3: UML-Diagramm des zugehörigen Panels zur Konfiguration (vereinfacht, vollständig in Abbildung A.3)

Die Toolib (Abbildung 4.2)ist für die direkte Kommunikation mit der Anwendung verantwortlich. Daher ist in diesem Fall in der Klasse CARMENlib die Steuerung von CARMEN und der Datenaustausch mit dem COM- Server implementiert. CARMENlib stammt von „ToollibFileSearchBase“ ab, damit im Dialog im festgelegten Pfad eine automatische Suche nach der CARMEN-Konfigurtionsdatei vorgenommen werden kann.

Die Methoden ”connectCarmen” und ”connectComServ” verbinden über COM die Anwendungen. Anschließend besteht die Möglichkeit mit ”loadConfiguration” in CARMEN

die gewünschte Konfiguration in die TopEd-Umgebung zu laden. Die Datei wird dabei als Parameter angegeben oder vorher mit ”setConfiguration” gesetzt werden. Beim Laden der Konfiguration wird auch die Testausführung vorbereitet, so dass Diese gegebenenfalls mit ”startCarmenTest” sofort gestartet werden kann. Man kann sie dann mit ”stopCarmenTest” auch wieder stoppen und gegebenenfalls CARMEN mit ”quitCarmen” beenden.

Die aufgeführten Methoden geben nach ihrer Durchführung das Ergebnis als Boolean-Wert zurück, so dass damit die Anweisungsabarbeitung ausgewertet werden kann. Die Toollib ist im Verzeichnis ToolLibs abzulegen, damit ECU-TEST bei der Suche nach Software das Vorhandensein von CARMEN erkennt. Damit auch das Konfigurationspanel angezeigt werden kann, muss die Datei CARMENlibPanel (Abbildung 4.3), in der das entsprechende Panel (Abbildung 4.4) definiert ist, sich ebenfalls im korrekten Verzeichnis befinden. ECU-TEST sucht bei jedem Start automatisch in diesen Verzeichnissen nach Toollibs und zugehörigen Dateien, damit die Schnittstellenvielfalt durch den Nutzer, mit dem nötigen Fachwissen und Programmüberblick, leicht erweitert werden kann.

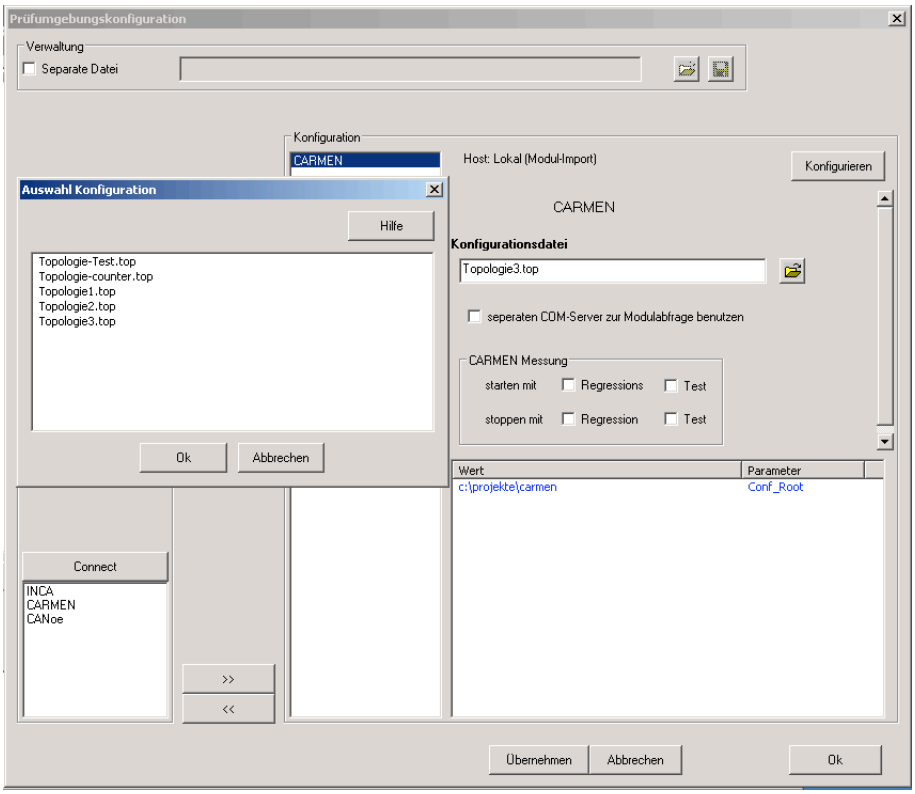

Abbildung 4.4: ECU-TEST - CARMEN Panel und entsprechende Topologie

In dem Panel ist die Variable ”Conf_Root” editierbar. Diese gibt an, ab welchem Verzeichnis nach einer gültigen CARMEN-Konfigurationen gesucht wird. Dies ist nötig, da

die Toollib auch über den Applikationsserver auf einem entfernten Testrechner nutzbar ist. In diesem Fall muss sich die Datei natürlich auf dem Rechner mit dem Applikationsserver befinden und wird entsprechend dort gesucht. Die Konfiguration kann durch drücken des Disketten- Buttons ausgewählt werden. Im Ergebniss wird auf dem lokalen oder entfernten Rechnern die Dateisuche begonnen und die Ergebnisse zur Auswahl angezeigt.

Außerdem wählt der Nutzer an dieser Stelle, in welcher Art und Weise die Verbindung zu ECU-TEST hergestellt wird, direkte Verbindung oder über den separaten COM-Server. Die unterschiedliche Handhabung ist in der Toollib selbst realisiert. Also wird für beide Varianten die gleiche Toollib verwendet und nur die Methoden verwenden teilweise unterschiedliche Anweisungen.

In der Konfiguration der Toollib besteht die Wahlmöglichkeit ob der CARMEN-Test automatisch beim Starten der ECU-TEST Regression oder des Tests gestartet beziehungweise gestoppt werden soll. Damit ist es möglich, die sonst nötigen Testschritte zum Starten oder Stoppen des Tests in CARMEN entfallen zu lassen.

Dieses Panel wird in der Datei Dialogs/Configuration/SoftwareTools/CARMENlibPanel.py definiert. In Abbildung 4.3 kann man erkennen, dass die Darstellung in der Klasse „SWToolDetailPanel“ vorgenommen wurde, die von „wxPanel“ erbt. Außerdem wird in „EntryCarmen“ der Aufruf des Suchdialogs vorgenommen.

4.2.2 BUSViewer

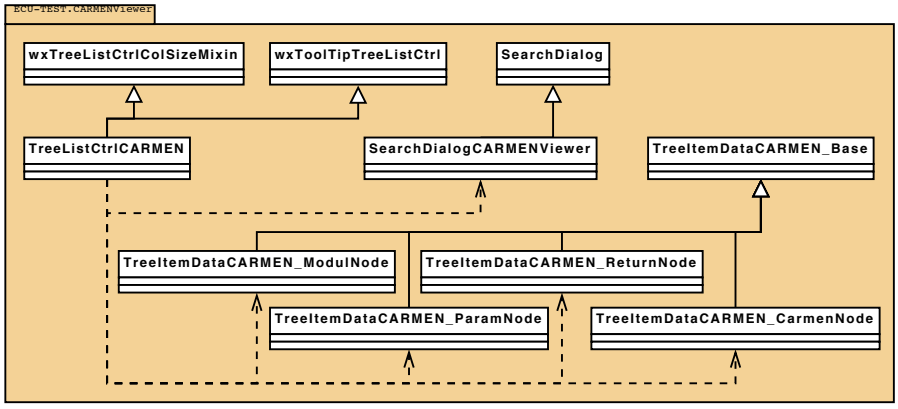

Abbildung 4.5: UML-Diagramm der für den BUSViewer nötigen Klassen (vereinfacht, vollständig in Abbildung A.6)

Im BUSViewer (Abbildung 4.5) wird dynamisch eine Liste der Module und ihrer Pa-

rameter und Rückgabewerte dargestellt, damit der Nutzer die Testschritte für sein Package erzeugen kann. Die eigentliche Liste ist in der Klasse TreeListCtrlCARMEN implementiert. Diese erbt natürlich über „wxTreeListCtrlColSizeMin“ alle nötigen Standartmethoden zur Darstellung als ausklappbarer Baum.

Damit, wie auch bei den anderen BUSViewern in ECU-TEST üblich, ein Durchsuchen durch Eingeben eines einzelnen Buchstabens möglich, ist nutzt der Viewer die „SearchDialogCARMENViewer“ Klasse, welche wiederum von „SearchDialog“ erbt. Diese zeigt dann den entsprechenden Dialog an und startet den Suchvorgang.

Beim Öffnen der Regression in ECU-TEST wird CARMEN gestartet und die gewählte Konfiguration geladen. Dadurch kann über die CARMENlib die Liste der vorhandenen Module geladen werden. Entweder haben diese Module sich im COM-Server registriert oder bieten über die entsprechenden Methoden an, welche Parameter und Rückgabewerte sie zur Kommunikation bereitstellen. Sollte die direkte Kommunikation gewählt worden sein, werden alle in der CARMEN-Konfiguration vorhandenen Module angezeigt. Auch solche die nicht direkt in der Lage sind mit ECU-TEST zu kommunizieren, sie sind dann selbstverständlich entsprechend markiert.

Da je nach Element des Baumes unterschiedliche Funktionen, also Testschritte eingefügt werden müssen, sind verschiedene Elemente in die wxTreeList eingehängt. Diese Elemente erben alle von „TreeItemDataCARMEN_Base“ und sind je nach ihrer Verwendung benannt. Das Startelement ist eine „TreeItemDataCARMEN_CarmenNode“. Mit Rechtsklick auf diese, kann ein CarmenControll Testschritt erzeugt werden, mit dem der CARMEN Test gestartet beziehungsweise gestoppt wird. Darunter befindet sich für jedes einzelne Modul in der CARMEN-Konfiguration eine einzelne „TreeItemDataCARMEN_ModulNode“, der die angebotenen „TreeItemDataCARMEN_ParamNode“ (Parameter lesen und Schreiben) und „TreeItemDataCARMEN_ReturnNode“ (Rückgabewerte Lesen) untergeordnet sind. Auch bei diesen Elementen bewirkt ein einfacher Rechtsklick die Auswahl des gewünschten Testschritts und das Einfügen in das Package.

4.2.3 Testklasse und CARMEN-Client

”CARMENclient”(Abbildung 4.6) ist im Grund ein Wrapper, der es ermöglicht die Toollib über den Applikationsserver zu betreiben, denn in dieser Klasse sind die SendCommand Methoden definiert, welche ihre Entsprechungen in der Toollib aufrufen. Daher ruft die Testklasse nicht direkt die Toollib-Methoden auf, sondern realisiert dies indirekt über

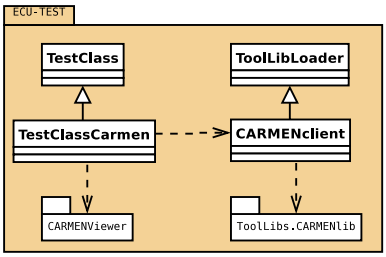

Abbildung 4.6: UML-Diagramm der Testklasse und CARMENclient-Klasse (vereinfacht, vollständig in Abbildung A.7)

”CARMENclient”.

Die Testklasse ”TestClassCarmen” ist für die Verwendung der Toollib in der Regression, der Testablaufsteuerung verantwortlich. Sie initialisiert die entsprechenden Testschritte, wertet die Daten bei der Durchführung aus und speichert die nötigen Daten zur Dokumentation. ”initCarmenAccess” initialisiert die Verbindung zu CARMEN und zum ECU-TEST-CARMEN-COM-Server, gegebenenfalls auch über den Applikationsserver.

In CARMEN wird ebenfalls die in der Softwarekonfiguration ausgewählte Datei geöffnet und in ”addModules” aus dem COM-Server die Liste der mit ECU-TEST kompatiblen Module ausgelesen und in die in ”CARMENViewer” definierte ”TreeListCtrl” zugefügt.

In der Liste werden alle Module und die angebotenen Parameter beziehungsweise Rückgabewerte angezeigt. Mit Rechtsklick können die spezifischen Testschritte eingefügt werden. Dabei wird in der Testklasse eine der folgenden fünf Methoden aufgerufen:

- ”NewTsControlCarmen” - CARMEN Test starten/stoppen
- ”NewTsReadReturn” - Rückgabewert lesen
- ”NewTsReadParam” - Parameter lesen
- ”NewTsWriteParam” - Parameter schreiben
- (”NewTsFuncCall” - Methode aufrufen)[1]

um eine Instanz der Testschrittklasse in den Testablauf einzufügen. Für nicht lesende

1 Wird nur bei direkten Verbindung mit CARMEN angeboten.

Testschritte wie ”TsWriteParam” oder ”TsControlCarmen” ist die Testdurchführung in der Methode ”OnRun” in der Testklasse implementiert. Diese ruft dann wiederum ”ReportControlCarmen” beziehungsweise ”ReportWriteParam” zum Dokumentieren der Ergebnisse auf. In ”Read” ist die Ausführung des lesenden Testschritts ”TsReadReturn” beschrieben, welche in ”DokuRead” dokumentiert wird.

Bei der direkten Anbindung an ECU-TEST über die Automatisierungsschnittstelle von CARMEN ist es zusätzlich möglich, beliebige Methoden von CARMEN-Modulen anzusprechen. Dafür gibt es in der Testklasse auch noch die Methode ”FuncCall”. Diese führt die angegebene Methode aus oder ruft den Wert eines Attributes ab.

Falls der Rückgabewert ein komplexer Typ, wie zum Beispiel eine Klasse ist, kann man eine Liste der Attribute angeben, deren Werte in einem Dictionary[1] organisiert zurück geliefert werden.

Da in den Testschritt-Klassen nur die Namen der Parameter beziehungsweise Rückgabewerte gespeichert sind, werden die Methoden ”getReadReturn”, ”getParamValue” und ”setParamValue” benötigt, um die Daten am korrekten Platz der Liste zu lesen oder zu schreiben.

4.2.4 Teststep-Klassen und zugehörige Konfigurationsdialoge

Die einzelnen Testschritt-Klassen („TsCallFunc“, “TsReadParam“, „TsReadReturn“, „TsControlCarmen“, „TsCallFunc“) erben entsprechend ihrer Funktionalität von „TsRead“, bei lesenden Testschritten, beziehungsweise von „TestStep“ und überschreiben dann die nötigen geerbten Methoden (Abbildung 4.7). Diese Klassen sind im Prinzip die Datencontainer für die Parameter der einzelnen Testschritte.

Die Klasse ”TsControlCarmen” ist zum Starten und Stoppen von Tests in CARMEN. Damit kann der Nutzer selbst entscheiden wie lange der CARMEN Test läuft. Entweder eine bestimmte Zeit oder abhängig von den gelesenen Daten.

Rückgabewerte können mit dem Testschritt ”TsReadReturn” gelesen werden. Natürlich muss das CARMEN Modul diese Daten auch im COM- Server speichern. Es ist auch möglich die Daten nach dem Stoppen des CARMEN- Tests weiterhin auszulesen. Mit

1 Python Datentyp einer Hashmap die aus Paaren von einem Schlüssel (Zahl oder String) und einem Wert (beliebiger einfacher oder komplexer Datentyp) besteht

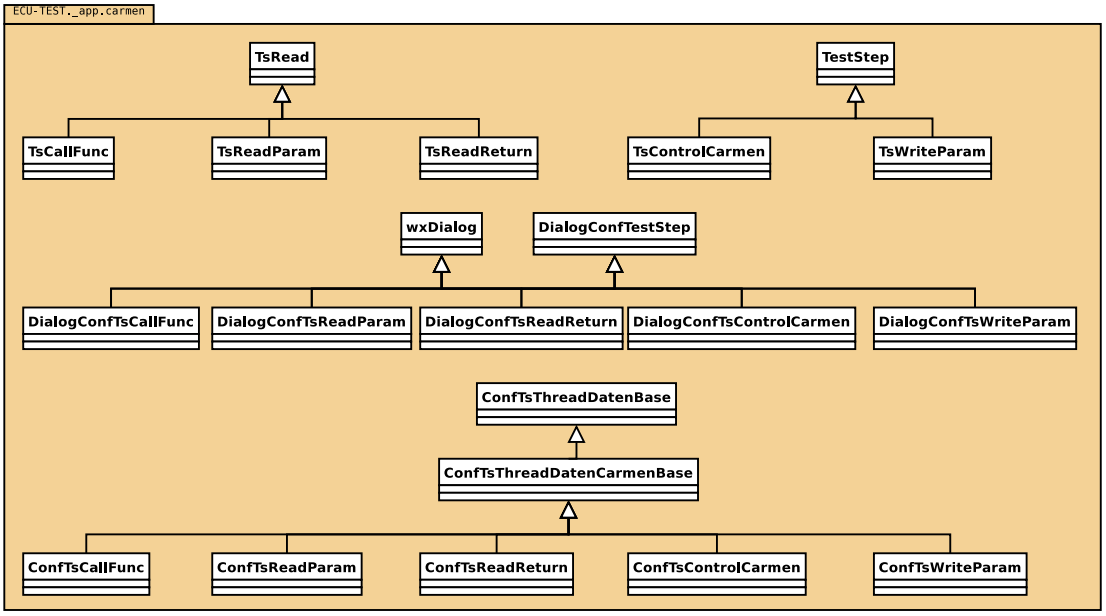

Abbildung 4.7: UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (vereinfacht, vollständig in Abbildung A.5 und A.4)

”TsWriteParam” werden Werte von dafür vorgesehenen Parametern gesetzt. In ECU-TEST besteht desweiteren die Möglichkeit diesen Testschritt während eines laufenden CARMEN-Tests auszuführen, allerdings ist es unwahrscheinlich, dass dies einen Einfluss auf den Testverlauf hat, da Parameter gewöhnlich nur beim Teststart vom Modul ausgelesen werden. Es ist auch möglich, falls der Bedarf entsteht, ein CARMEN- Modul zu Programmieren das dies anders handhabt.

Beim Einfügen der Testschritte. beziehungsweise beim nachträglichen Editieren werden die Dialoge angezeigt, die in ”DialogConfTsControlCarmen”, ”DialogConfTsReadReturn”, ”DialogConfTsReadParam”(Abbildung 4.8), ”DialogConfTsWriteParam”(Abbildung 4.9) und DialogConfTsCallFunc” definiert sind. Beim Einfügen des Testschritts zum Lesen der Rückgabewerte, wird der Standart ECU-TEST Dialog „DialogConfTeststep“ zum Definieren der Bedingungen genutzt, deshalb stammen alle Dialog Klassen von dieser ab und ergänzen das Fenster mit den nötigen Eingabefeldern. Damit ist es möglich, komplexe Bedingungen für die Auswertung der Rückgabewerte und Parameter festzulegen.

Zum Aufrufen dieser Dialoge sind die Wrapper-Klassen „ConfTsCallFunc“, „ConfTsReadParam“, „ConfTsReadReturn“, „ConfTsControlCarmen“ und „ConfTsCallFunc“ implementiert. Sie erstellen die korrespondierenden Dialoge und verknüpfen sie mit Daten durch die Teststep-Klasse damit die korrekten Daten angezeigt und verändert werden können.

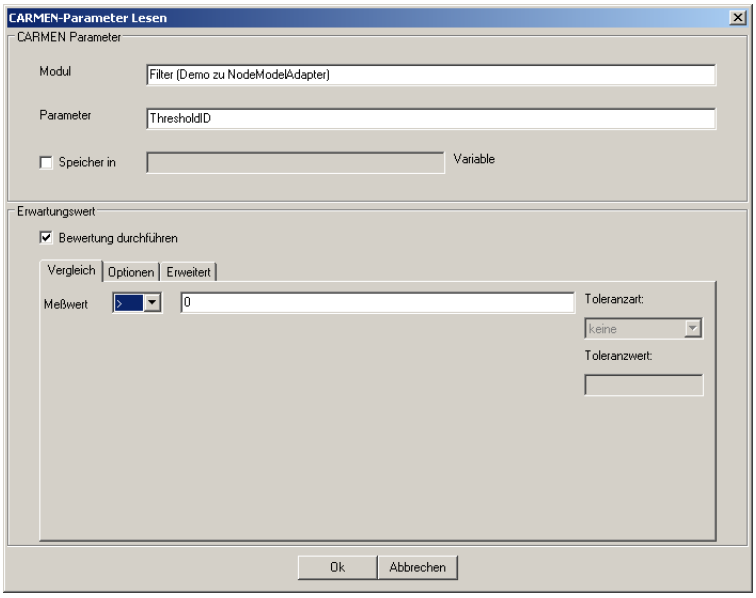

Abbildung 4.8: Testschrittdialog - Parameter lesen

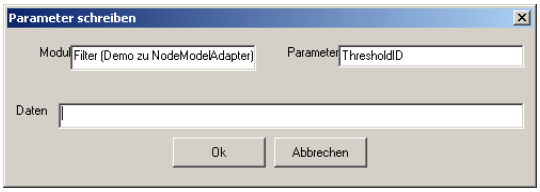

Abbildung 4.9: Testschrittdialog - Parameter schreiben

4.2.5 Testablauf

Da CARMEN nicht wie ECU-TEST sequentiell sondern eher ereignisorientiert arbeitet, muss der Testablauf entsprechend angelegt werden. Somit werden zu Beginn des ECU-TEST Testablaufs (Abbildung 4.10) die Parameter des CARMEN Moduls gesetzt, bis alle Einstellungen vorgenommen sind. Danach wird der TEST in CARMEN gestartet und je nach Ziel des Test die nötige Zeit gewartet, damit innerhalb von CARMEN genügend Nachrichten in das Modul fließen können. Während der Wartezeit kann man auch regelmäßig die Daten aus dem Modul auslesen und die Zeitspanne anhand dieser Daten anpassen, wenn man zum Beispiel auf einen speziellen Fehler wartet. Nachdem der Test in CARMEN gestoppt wurde, können erneut die Rückgabewerte ausgelesen werden.

Damit ist eine einfache Testsequenz in etwa folgendermaßen aufgebaut (Abbildung 4.10):

1. Setzen der Parameter ”TsWriteParam”

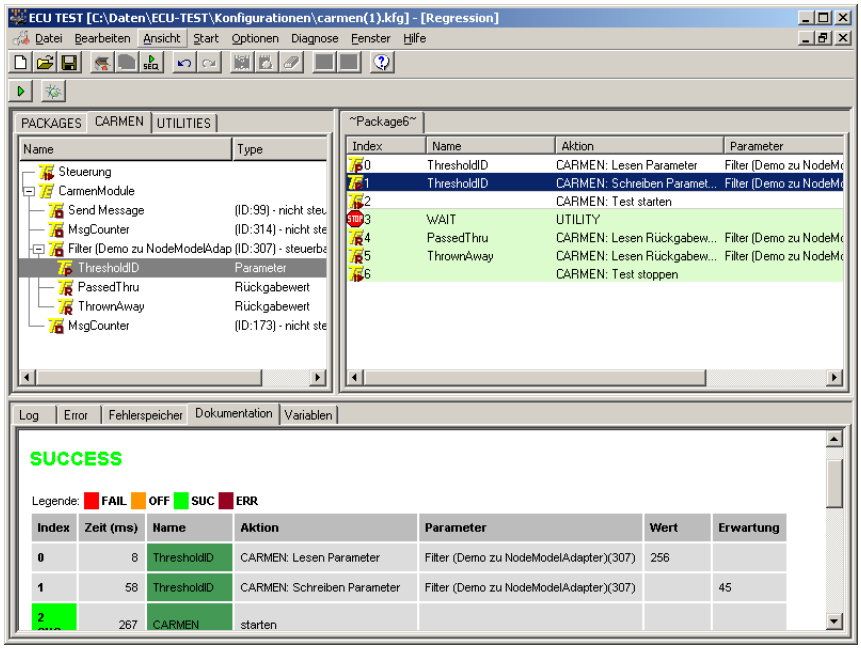

Abbildung 4.10: ECU-TEST - typische Testsequenz

2. Starten des CARMEN-Tests ”TsControlCarmen”

3. Ein Warten-Testschritt oder andere Schritte, die ein genügend große Zeit verschreiten lassen, um eine ausreichende Menge Daten in CARMEN zu sammeln

4. Auslesen der Rückgabewerte ”TsReadReturn”

5. Stoppen des Tests ”TsControlCarmen”, gegebenenfalls können auch die oberen beiden Testschritte in einer Schleife wiederholt werden.

6. Finales auslesen der Rückgabewerte ”TsReadReturn”

Wenn bei der Konfiguration der Toollib angegeben wurde, dass der Test in CARMEN automatisch mit der Regression oder dem ECU-TEST Test gestartet und gestoppt werden soll, können diese Schritte dann natürlich entfallen.

4.3 CARMEN-Modul

Das CARMEN- Modul muss der von CARMEN vorgegebenen API entsprechen. Zusätzlich dazu ist es erforderlich, dass die Anweisungen zur Registrierung und Datenaustausch

mit dem COM-Server implementiert werden. Bei der direkten Anbindung ist diese Registrierung selbstverständlich nicht notwendig, allerdings erwartet ECU-TEST spezielle Methoden, um die angebotenen Parameter, Rückgabewerte und Methoden-Namen zu ermitteln.

Bei der Initialisierung des CARMEN -Moduls (”OnInit”) muss dieses eine Verbindung zum COM- Server aufbauen und gleichzeitig ist eine Registrierung des Moduls dort notwendig. Damit auf die Rückgabewerte und Parameter zugegriffen werden kann, sind deren Namen auch an den COM-Server zu übermitteln.

```
Private Function IStandardModule_OnInit(ByVal progressReport As Object)
    As Boolean

    ' True: kein Fehler in Initialisierung
    IStandardModule_OnInit = True

    ' Mit ECUTEST-ComServer verbinden
    Set comserver = CreateObject("ECUTestCarmenCOM.Application")
    ' registriert das modul und speichert die ID
    comID = comserver.registeringModule(IStandardModule_Name())
    ' registiert die Rueckgabewerte und Parameter
    comserver.addReturnName ("lost")
    comserver.addReturnName ("found")
    comserver.addParamName ("MaxMsgToSearch")
End Function
```

Des Weiteren sollte das Modul dem COM- Server übermitteln, ob ein Testvorgang läuft oder nicht. Dafür bietet der COM- Server die Methode ”setRunning”, welche einen Booleschen Wert als Parameter erwartet und vorzugsweise in der Start- (”OnPreStart”) beziehungsweise Stop- Methode (”OnStop”) aufgerufen werden sollte. Zusätzlich müssen vor dem Teststart alle Parameter die von außen gesetzt werden können, ausgelesen werden. Dies geschieht optimaler Weise ebenfalls in der Methode ”OnPreStart”.

```
' Parameterdaten Auslesen
    m_MaxMsgToSearch = Val(comserver.getParamValue(0))
```

Die Datenübermittlung kann nach eigenen Vorstellungen realisiert werden. Im implementierten Modul wurde ein Timer gestartet und beim entsprechenden Event das Senden der Daten initiiert.

```
Private Sub IStandardModule_OnTimer(ByVal timerId As Long)
    Dim mydata As Variant
.....
```

```
    ' schickt die Daten zum Comserver
    mydata = Array(m_lostMsg, m_foundMsg)
    comserver.setReturnValues (mydata)
End Sub
```

Selbstverständlich ist es auch denkbar, die Daten erst zum Ende des Tests in der Methode ”OnStop” zu übermitteln. Auf alle Fälle sollte selbst bei regelmäßigen Setzen der Daten ebenso am Ende des Tests noch einmal eine Aktualisierung sichergestellt werden, so dass der COM- Server den aktuellsten Stand besitzt.

Problematisch ist das mehrfache Vorhandensein eines Moduls in CARMEN. Zur eindeutigen Identifizierung im COM- Server werden bei der Registrierung im Server jedem Modul eine ID zurückgeliefert. Der Programmierer des CARMEN- Moduls sollte eine Anzeige dieser ID ermöglichen, damit in ECU-TEST bei der Testerstellung die korrekten Module zugeordnet werden. Da die Zuteilung durch die Reihenfolge bei der Initialisierung der Module bestimmt wird, ist es sinnvoll die Konfiguration, obwohl es möglich wäre, nicht beim laufenden ECU-TEST zu ändern.

Zwar kann die Auflistung in ECU-TEST aktualisiert werden, aber die eindeutige Identifizierung in ECU-TEST erfolgt ebenfalls über die ID und kann daher beim erneuten Öffnen der Konfiguration abweichen. Im normalen Betrieb sollte dies kein großes Problem darstellen, da die Auswertung in ECU-TEST erst nach Fertigstellung der CARMEN-Konfiguration vollzogen wird.

Zur direkten Anbindung des Moduls über die CARMEN eigene Automatisierungsschnittstelle erwartet ECU-TEST die Properties ”ParamNames”, ”ReturnNames” und ”MethodNames”. Diese liefern jeweils eine Liste der Namen, der Parameter, Rückgabewerte oder Methoden. Wenn in den Properties nur ein einzelner String enthalten ist, so enthält er eine durch Semikolon getrennte Auflistung der entsprechenden Namen.

Diese drei Properties und die in ihnen bezeichneten Methoden und Eigenschaften müssen natürlich über die COM-Schnittstelle publiziert werden, um auf sie zugreifen zu können. Selbstverständlich registrieren die Programmbausteine sich selbst in Windows, wodurch alle Methoden und Attribute aus der Typelib gelesen werden können.

Das ist leicht zu erkennen wenn man mit einem üblichen COM -Browser (zum Beispiel ”combrows” - ist beim Python win Paket enthalten oder OLE/COM Object Viewer von Microsoft Visual) einen Blick über die im System registrierten Bibliotheken wirft (Abbildung 4.11). Die Bibliothek kann mit ”makepy.py” in ein Python Script umgewandelt werden, welches es nach der Einbeziehung in ECU-TEST auch ermöglicht, dass die

Klassen beim COM Zugriff erkannt werden und über Python interne Funktionen das automatische Ermitteln aller angebotenen Methoden und Attribute verfügbar macht.

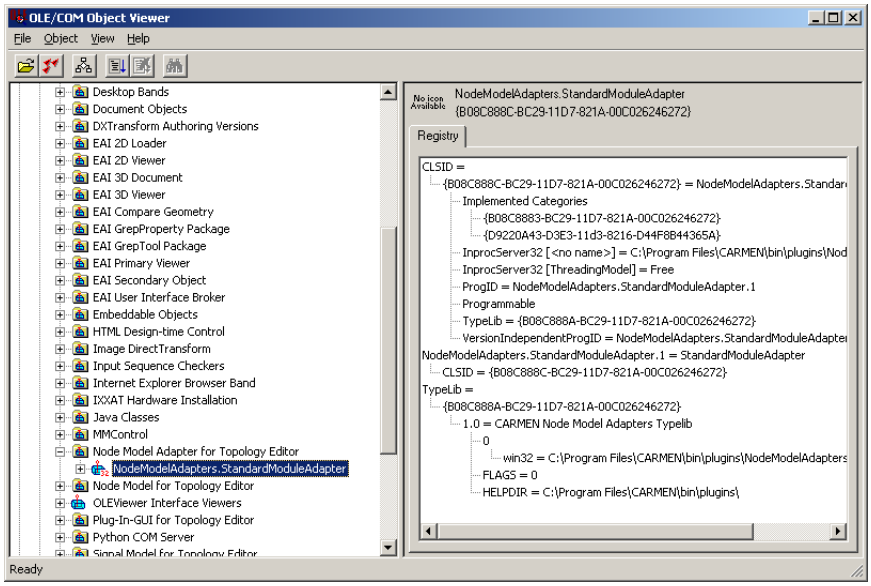

Abbildung 4.11: COM Object Viewer

Das Problem dieser Methode liegt im Endergebnis, da der Browser sozusagen raten muss, als was die Attribute gedacht sind - als Parameter oder Rückgabewerte. Dabei verallgemeinert ECU-TEST alle Properties, auf die nur ein Lesezugriff möglich ist als Rückgabewert und alle die zumindest schreibbar sind, als Parameter. Bei den Methoden gibt es keine Unterscheidung.

Als zusätzliches Feature kann eine Property im Modul implementiert werden, deren Namen mit ”help_” beginnt und danach den Namen eines bereits vorhandenen Attributs oder Methode trägt. Diese Properties enthalten einen String der in ECU-TEST als Tooltip angezeigt wird, wenn man mit der Maus über das Element fährt.

In CARMEN wurde nur ein Beispiel-Modul implementiert um zu testen, ob die Schnittstellen vollständig funktionieren. Dazu wurde aufgrund der Einfachheit Visual-Basic© benutzt. Für diese Programmiersprache bietet CARMEN ein vorbereitetes Modul den ”NodeModelAdapter”. Dieser realisiert einen einfachen Paketzähler, der die Anzahl der Pakete aufsummiert, die eine vorgegebene Nachrichten-ID überschreiten. Das Ziel war es, dass von ECU-TEST die spezifische ID ab wann eine Nachricht gezählt wird (Parameter) und die Anzahl der Nachrichten (Rückgabewert) ausgelesen beziehungsweise geschrieben werden kann.

Das CARMEN Modul implementiert die Schnittstelle, die durch die Klasse ”IStandart-

Module" vorgegebenen werden. Dazu zählen die Methoden "OnStart" und "PreStart", um die Initialisierung zu Beginn des CARMEN-Teststarts vorzunehmen. "OnStop" wird natürlich nach dem Beenden des CARMEN-Tests aufgerufen und arbeitet entsprechende Anweisungen ab. Die Nachrichtenverarbeitung erfolgt wie schon erwähnt Event basiert. Wenn also die Nachricht am Eingangsport ankommt, wird die Methode "OnMessage" aufgerufen. Diese enthält als Argument das Message-Objekt, welches je nachdem von welchem Bus es kommt, unterschiedliche Methoden/Attribute enthält. Der Parameter Port bezeichnet den Port, an dem die Nachricht empfangen wurde.

Sollten Timer definiert worden sein, so wird identisch zu "OnMessage" die Methode "OnTimer" aufgerufen, wenn das Event getriggert wird. Die ID ist dann die bei der Timererstellung vergebene Kennnummer.

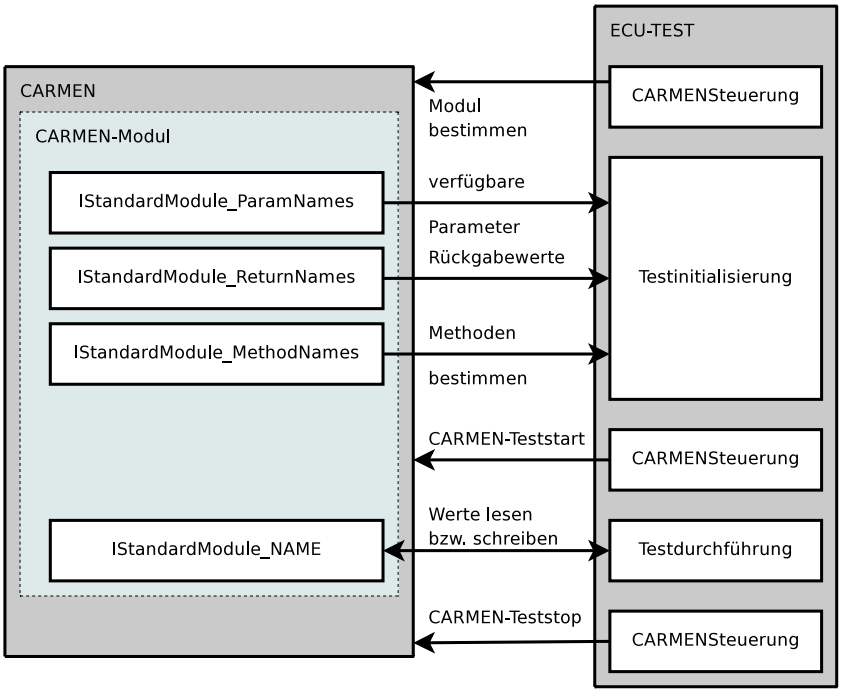

Abbildung 4.12: Funktionsweise des CARMEN-Moduls bei der direkten Anbindung an ECU-TEST

Für die Kommunikation bei der Variante 2, also der direkten Implementation, ist es aus Sicht des Programmierers nichts weiter nötig, als die Attribute und Methoden öffentlich ("public") zu definieren. In Visual-Basic wird bei der Projekterstellung, also dem Compilieren zur ocx Datei automatisch alles erzeugt was erforderlich ist, die Typebibliothek korrekt zu registrieren und alle öffentlichen Attribute/Methoden über COM aufrufbar sind (Abbildung 4.12). Sollte das Modul in C++ implementiert sein, so muss natürlich die entsprechende IDL Datei und Typelib erzeugt werden. Damit allerdings eine struk-

turierte Anzeige in ECU-TEST erfolgen kann, hat der Programmierer die Attribute ”ParamNames”, ”ReturnNames” und ”MethodNames” zu implementieren. Diese geben entweder eine Liste von Strings der entsprechenden Namen oder einen einzelnen String, der aus allen Namen getrennt durch ein Semikolon besteht.

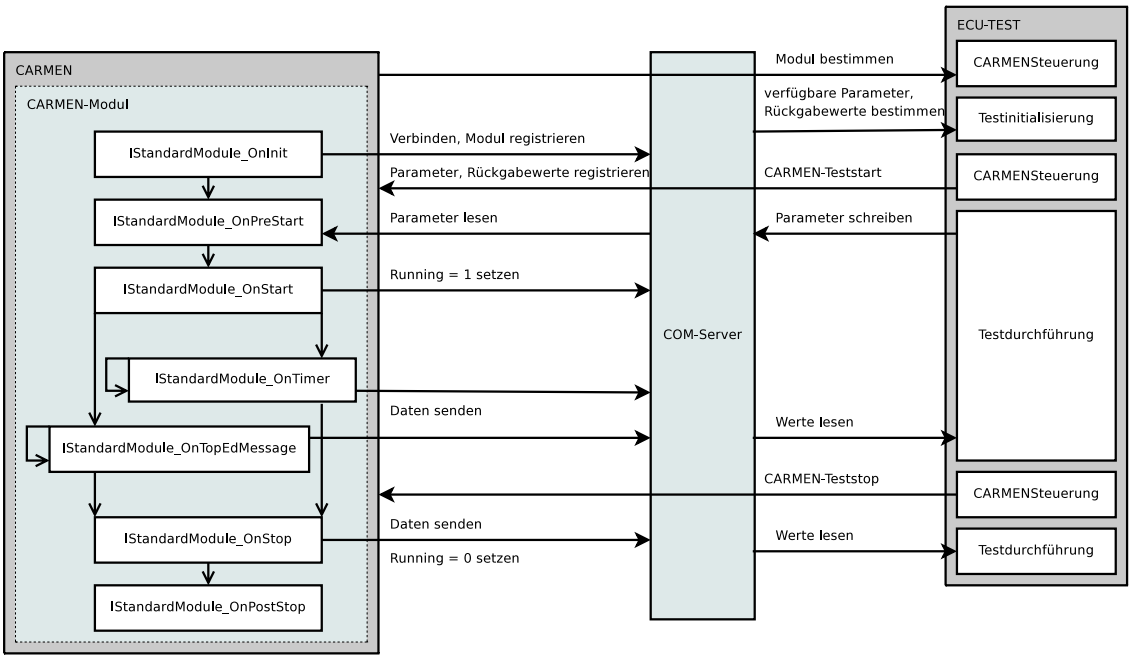

Abbildung 4.13: Funktionsweise des CARMEN-Moduls bei der indirekten Anbindung, über den COM-Server, an ECU-TEST

Bei der Variante 1, der Kommunikation über den eigenen COM-Server hat das CARMEN Modul aktiv dafür zu sorgen, dass immer aktuelle Daten im Server abgelegt sind (Abbildung 4.13). Dazu eignen sich im Grunde zwei Events:

- Timer-Events
- Nachrichten-Events

Werden nur bei wenigen Nachrichten in unregelmäßigen Zeitabständen Daten erzeugt, so ist es ratsam, diese direkt nach dem Entstehen zum COM-Server zu senden. Allerdings ist der COM-Zugriff im Verhältnis zum Rest der Prozesses sehr langsam, daher ist es möglich, dass dieser Zeitpunkt für die Datenablage zur Blockierung des Moduls führt.

Wenn es sich um Daten handelt die oft durch viele Nachrichten aktualisiert werden, eignet sich eher die Einrichtung eines Timers, um die Aktualisierung im COM-Server vorzunehmen. Die Aufruffrequenz sollte entsprechend des Bedarfs gewählt werden. Natürlich schließen sich beide Möglichkeiten nur bedingt aus, da ein Modul gegebenenfalls

unterschiedliche Nachrichtentypen verarbeitet.

4.4 Schlussfolgerung

Abschließend kann man feststellen, dass eine Schnittstelle zwischen CARMEN und ECU-TEST mit einigen wenigen Einschränkungen realisiert und in der Praxis nutzbar ist. Desweitern hat sich, wie vorhergesehen, die direkte Variante als die bessere Lösung herausgestellt. Für diese Umsetzung sprechen nachfolgende Vorteile:

- direkte Datenbeschaffung zum gewählten Zeitpunkt im ECU-TEST und damit genaue Werte zum Lesezeitpunkt.
- die Möglichkeit des dynamischen Datenhandlings
- einfacheres Handling und höhere Betriebssicherheit durch den Verzicht auf einen Zwischenprozess (ECU-TEST-CARMEN-COM-Server)
- keine zusätzliche Installation des COM-Servers nötig

Im Speziellen ist die Möglichkeit, sogar Klassen zur Datenhaltung als Rückgabewert einer Methode zu deklarieren, hervorzuheben.

Die indirekte Datenübergabe wird zwar aus Kompatibilitätsgründen noch für eine Weile in ECU-TEST erhalten bleiben, aber in naher Zukunft aufgrund der Nachteile nicht weiterentwickelt werden.

5 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde eine Schnittstelle zwischen dem Testautomatisierungswerkzeug ECU-TEST, das im Bereich automatisierter Softwaretest von Steuergeräten eingesetzt wird und dem Busanalysewerkzeug CARMEN, das für Bussysteme im Kraftfahrzeugumfeld entwickelt wurde, implementiert. Mit Hilfe dieser Schnittstelle ist es möglich, die nötigen Funktionen in CARMEN und nach der Vorgabe angepasste CARMEN-Module zu steuern.

Zunächst wurden einige Grundlagen zur verwendeten Kommunikation über die COM-Schnittstelle und der verwendeten Programmiersprache Python näher erläutert.

Nachdem die daraus folgenden Realisierungsvarianten der Anbindung entworfen und die Architektur von ECU-TEST beschrieben wurde, war es nötig auf die nötigen Motifikationen zur Integration der neuen Schnittstelle einzugehen. Im weiteren wurde der Aufbau eines Moduls in CARMEN skizziert. Dabei sind verschiedene Möglichkeiten zur Verwendung der Schnittstelle dargestellt und ihre Vorteile, beziehungsweise Probleme diskutiert worden.

Danach wurde die Implementierung der entworfenen Elemente für beide Realisierungsvarianten beschrieben und der Aufbau eines einfachen Moduls zum Testen der Schnittstelle gezeigt. In der Schlussfolgerung wurden die beiden Realisierungsvarianten gegenübergestellt und ihre Eignung bewertet.

Zukünftig ist es nun nötig Module zu erstellen, die diese Schnittstelle nutzen, wodurch dann je nach Komplexität und Aufgabenfeld der Module, weitere Anforderungen der Schnittstelle entstehen könnten. Außerdem wird eine Portierung auf ECU-TEST 4 stattfinden müssen, damit das Buswerkzeug CARMEN auch damit genutzt werden kann.

Literaturverzeichnis

[Gmb] GMBH, TraceTronic: *Homepage TraceTronicGmbH (www.tracetronic.de)*. http://www.tracetronic.de

[Hau99] HAUPT, Horst F.: *Visual Basic Referenz*. Franzis-Verlag GmbH, 1999 (3-7723-7233-3)

[Ing] Ingenieurbüro Matthias Puchner: *Anwendungshilfe zu Topologie Editor*

[Loo01] LOOS, Peter: *Go To COM*. Addison-Wesley, 2001 (3-8273-1678-2)

[Lut99] LUTZ, Mark: *Python kurz und gut (Taschenbuch)*. O'Reilly, 1999 (3-89721-511-X)

[MN05] MARKUS NIX, Torsten Marek Michael Weigend Wolfgang W.: *Exploring Python*. entwickler.press, 2005 (3-935042-69-8)

[Oes05] OESTEREICH, Bernd: *Analyse und Design mit UML 2*. Oldenbourg, 2005 (3-486-57654-2)

[Pyt] *Python 2.5 Documentation*. http://docs.python.org/download.html

[Wei05] WEIGEND, Michael: *Objektorientierte Programmierung mit Python*. mitp-Verlag, 2005 (3-8266-1571-9)

Abbildungsverzeichnis

1.1 ECU-TEST Anwendung 3
1.2 CARMEN - TopEd Anwendung 4

2.1 COM Kommunikation 6
2.2 Kommunikation ohne Events 8
2.3 Kommunikation mit Events 8
2.4 Inproc-Server 9
2.5 lokaler Server 9
2.6 Beispiel eines Funktionsaufruf in Python mit if-then Block 10
2.7 Beispiel eines COM-Servers in Python 12

3.1 indirekter Datenaustausch mit separaten COM-Server 14
3.2 direkter Datenaustausch mit CARMEN 15
3.3 ECU-TEST - Architektur 16
3.4 ECU-TEST - Regression 17
3.5 Nachrichtenüberprüfung in ECU-TEST 20
3.6 Nachrichtenüberprüfung in CARMEN 21

4.1 UML-Diagramm des ECU-TEST-CARMEN-COM-Server (vereinfacht, vollständig in Abbildung A.1) 22
4.2 UML-Diagramm der Toollib (vereinfacht, vollständig in Abbildung A.2) . 25
4.3 UML-Diagramm des zugehörigen Panels zur Konfiguration (vereinfacht, vollständig in Abbildung A.3) 25
4.4 ECU-TEST - CARMEN Panel und entsprechende Topologie 26
4.5 UML-Diagramm der für den BUSViewer nötigen Klassen (vereinfacht, vollständig in Abbildung A.6) 27
4.6 UML-Diagramm der Testklasse und CARMENclient-Klasse (vereinfacht, vollständig in Abbildung A.7) 29
4.7 UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (vereinfacht, vollständig in Abbildung A.5 und A.4) 31
4.8 Testschrittdialog - Parameter lesen 32
4.9 Testschrittdialog - Parameter schreiben 32

4.10 ECU-TEST - typische Testsequenz 33
4.11 COM Object Viewer 36
4.12 Funktionsweise des CARMEN-Moduls bei der direkten Anbindung an ECU-TEST 37
4.13 Funktionsweise des CARMEN-Moduls bei der indirekten Anbindung, über den COM-Server, an ECU-TEST 38

A.1 UML-Diagramm des ECU-TEST-CARMEN-COM-Server 44
A.2 UML-Diagramm der Toollib 45
A.3 UML-Diagramm des zugehörigen Panels zur Konfiguration 46
A.4 UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (Teil 1) 47
A.5 UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (Teil 2) 48
A.6 UML-Diagramm der für den BUSViewer nötigen Klassen 49
A.7 UML-Diagramm der Testklasse und CARMENclient-Klasse 50

A UML Diagramme

A.1 COM-Server

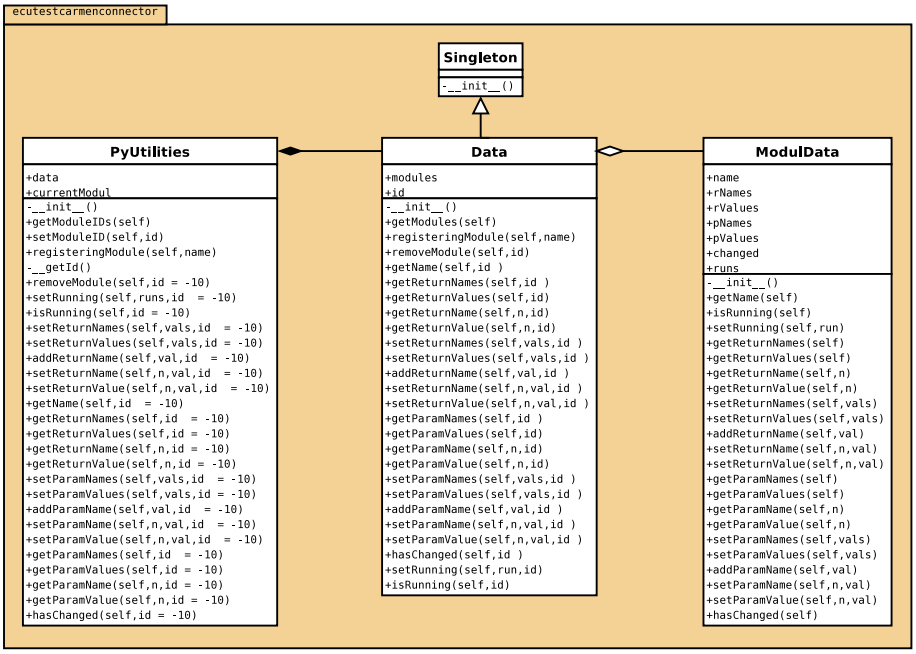

Abbildung A.1: UML-Diagramm des ECU-TEST-CARMEN-COM-Server

A.2 Toollib

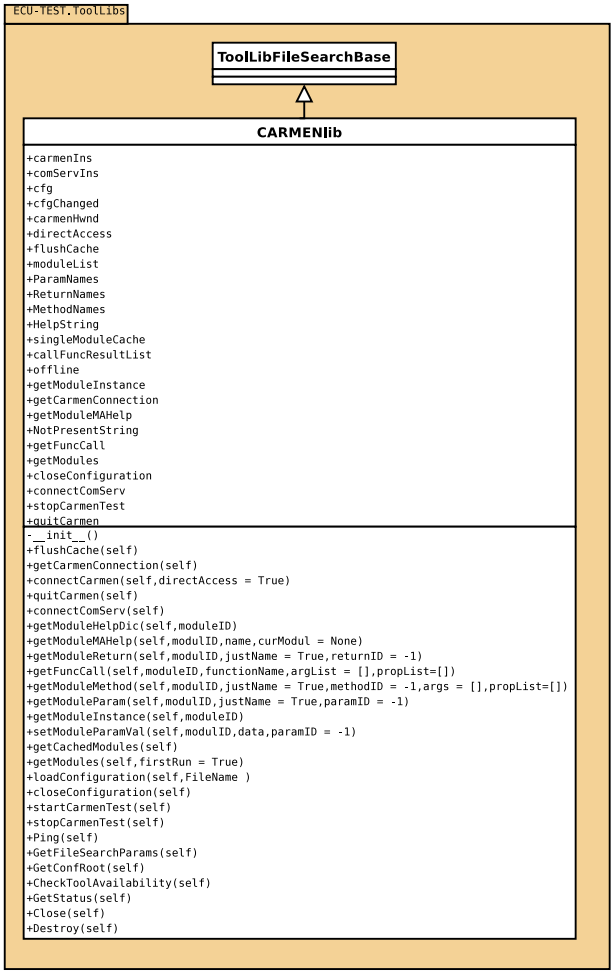

Abbildung A.2: UML-Diagramm der Toollib

A.3 Konfigurationspanel

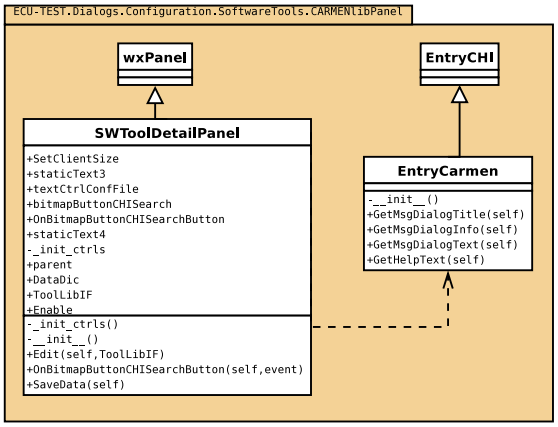

Abbildung A.3: UML-Diagramm des zugehörigen Panels zur Konfiguration

A.4 Teststep-Container

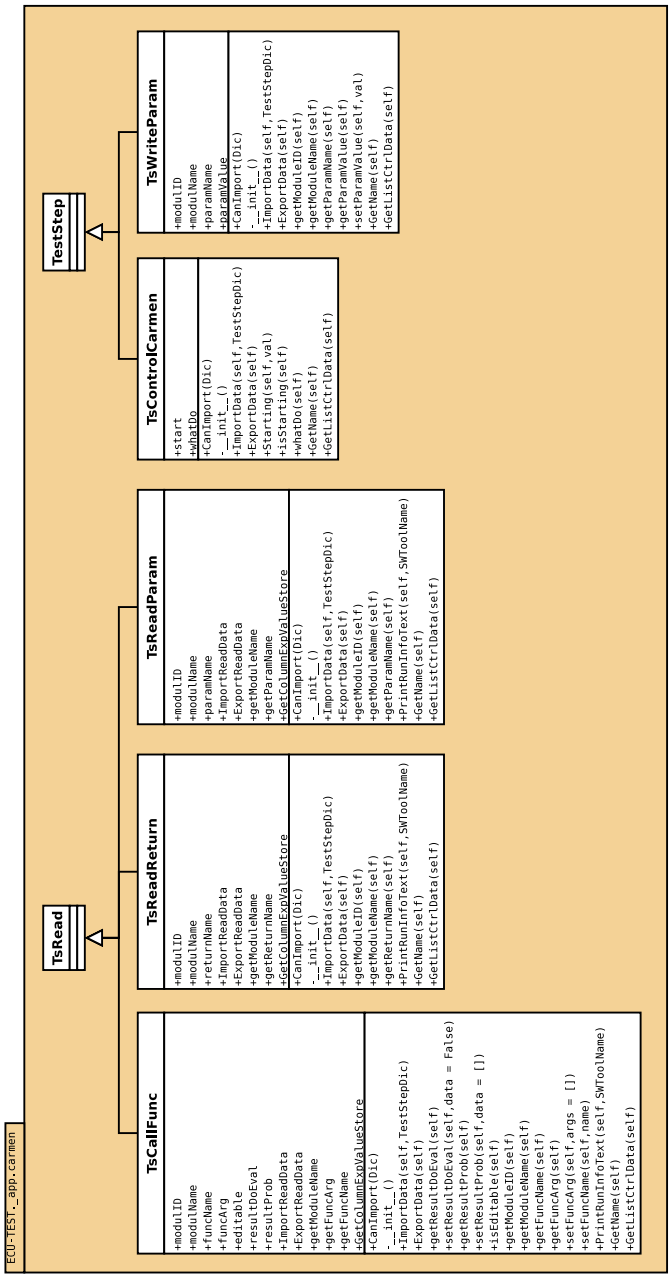

Abbildung A.4: UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (Teil 1)

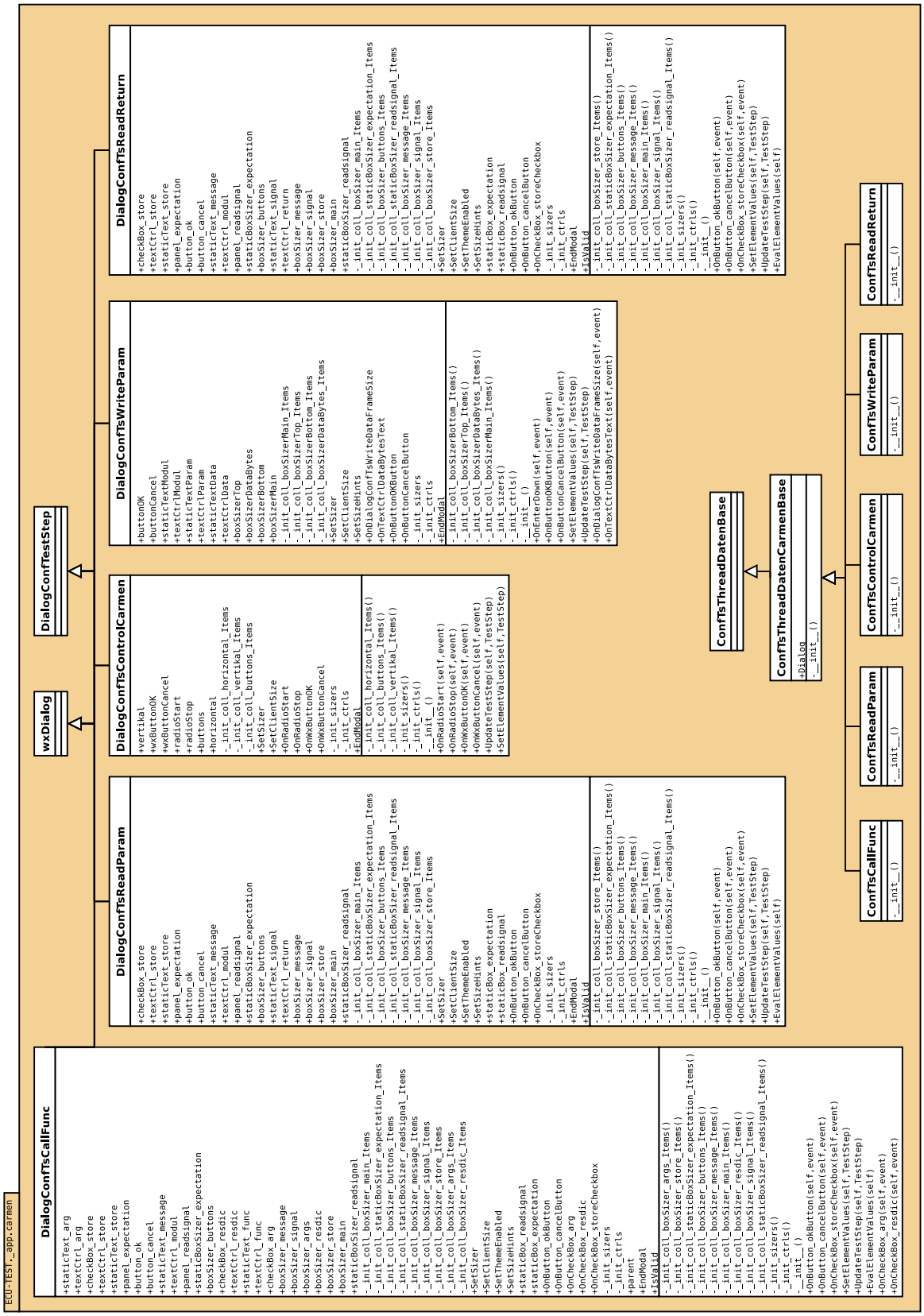

Abbildung A.5: UML-Diagramm der Teststep-Klassen und Konfigurationsdialoge (Teil 2)

A.5 Busviewer

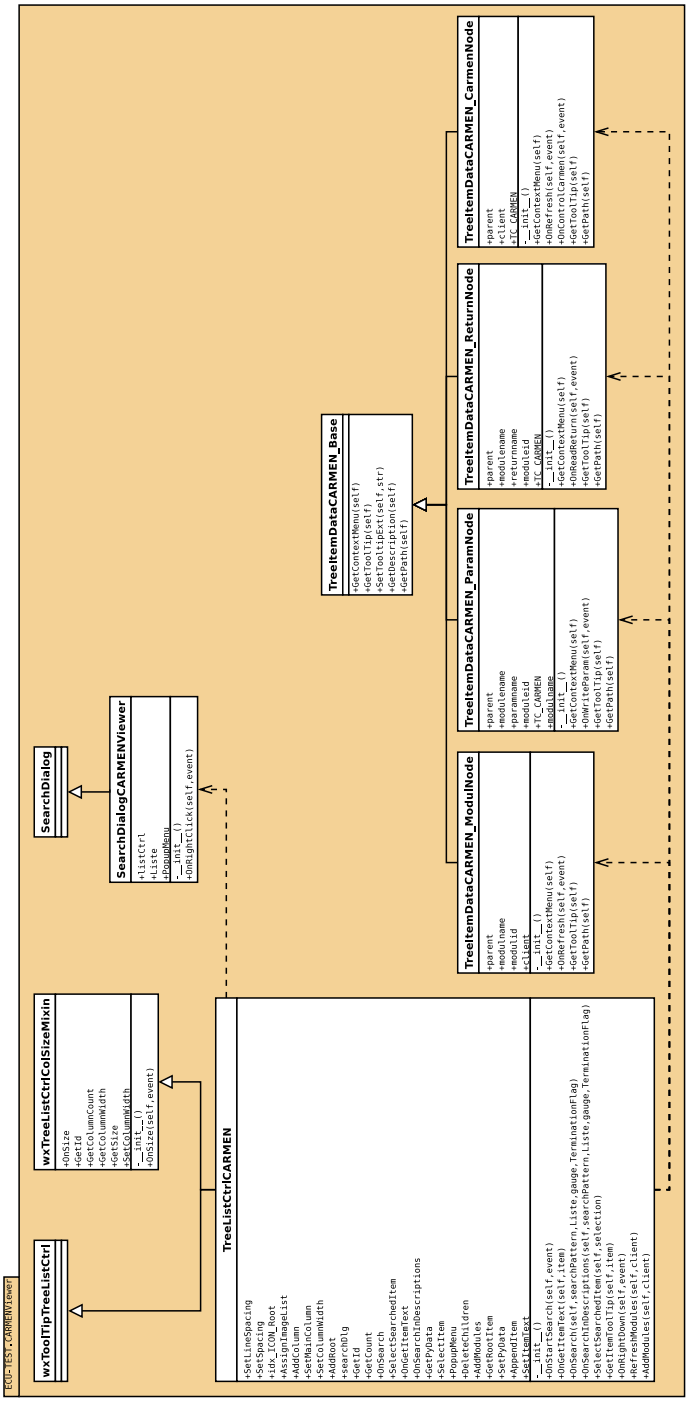

Abbildung A.6: UML-Diagramm der für den BUSViewer nötigen Klassen

A.6 Integration in die Regression

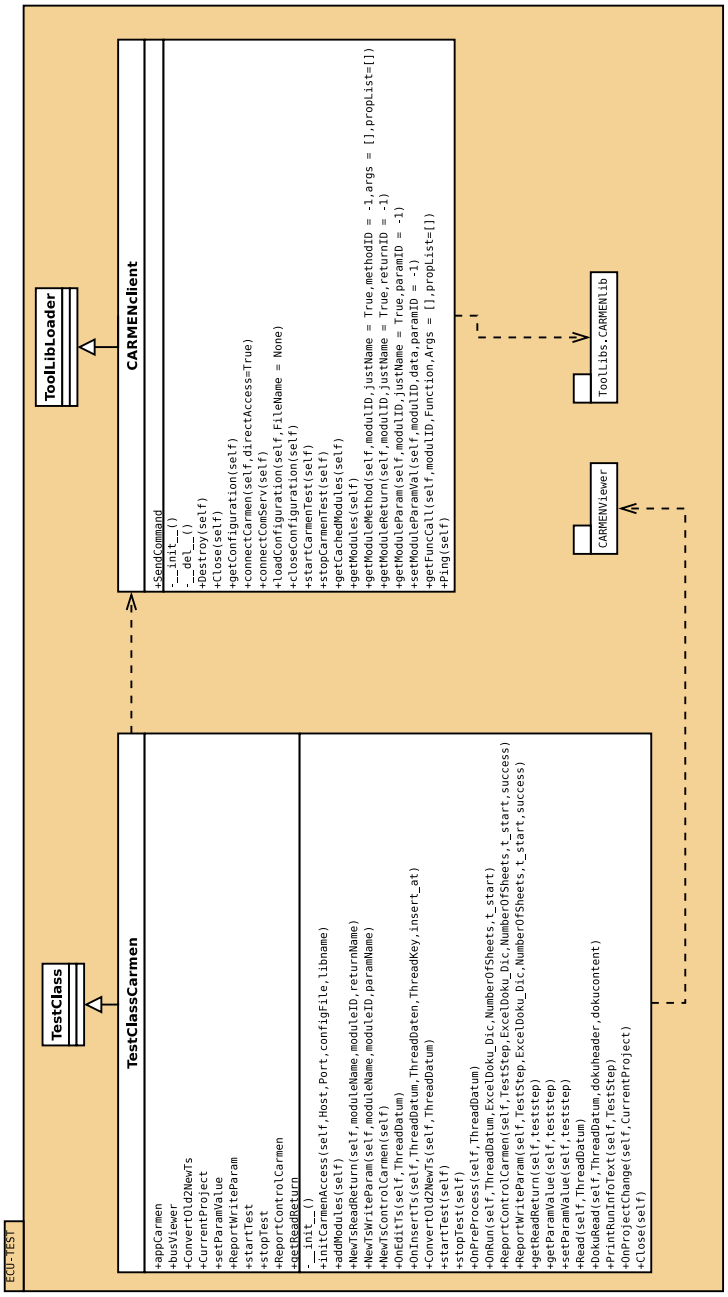

Abbildung A.7: UML-Diagramm der Testklasse und CARMENclient-Klasse