TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Angewandte Informatik

Diplomarbeit

Entwicklung eines generischen Architekturkonzeptes zur Integration von Traceanalysekomponenten und Messwerkzeugen in eine Testautomatisierungsumgebung

Bearbeiter: Sven Richter

Betreuer: Dr. Christian Pigorsch (BMW Group)
Dr.-Ing. Rocco Deutschman (TraceTronic GmbH)
Dipl.-Inf. Andreas Richter (TU-Dresden)

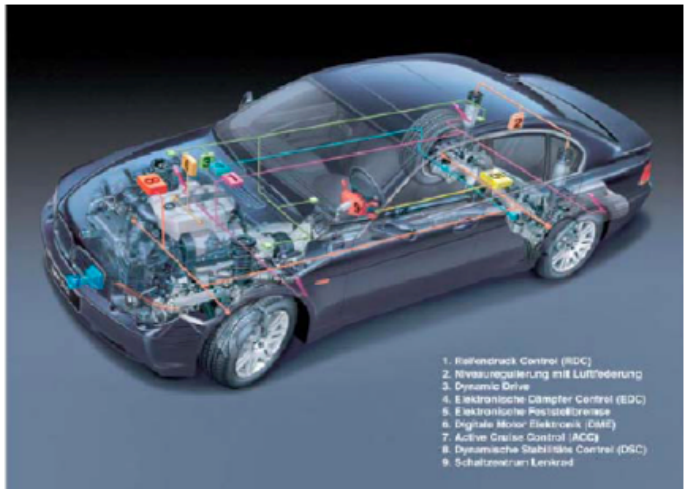
Datum, Ort: 27.03.2009, Dresden

Gliederung

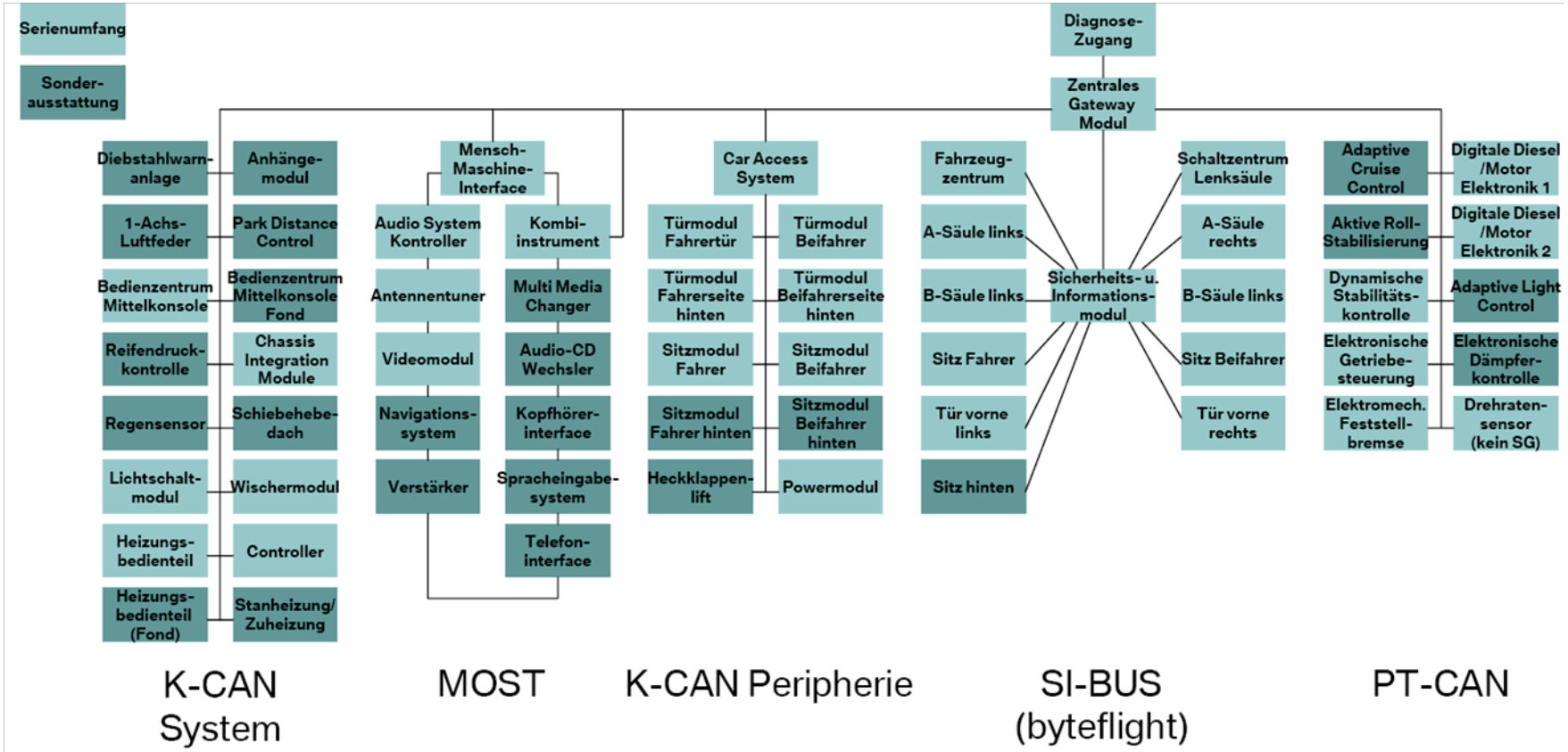
- Motivation und Ziel der Arbeit
- Automatenschnittstelle
- Zertifizierungsanwendung und Zählautomat
- CARMEN-Modul
- Python-Automat-Adapter
- Anwendungsfall
- Zusammenfassung und Ausblick

Moderne Kraftfahrzeugelektronik

- dominiert durch Steuergeräte und Busse (CAN, FlexRay, MOST)
- verschiedene Bustypen und -geschwindigkeiten, je nach Anwendung (Antriebsstrang, Karosserieelektronik etc.)
- Premiumfahrzeug ca. 80 Steuergeräte (2500 software-gestützte Funktionen)
- in Zukunft steigende Komplexität und erhöhte Abhängigkeiten



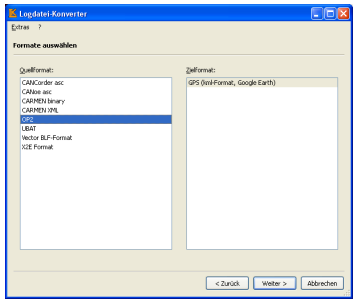
Motivation und Ziel der Arbeit



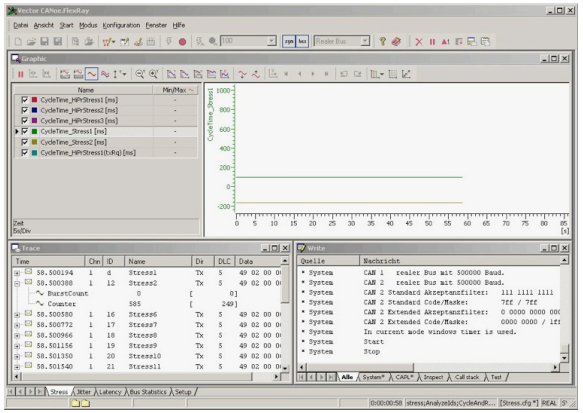
[Quelle: Vorlesung Softwareentwicklung und –absicherung, TraceTronic GmbH]

Werkzeuge zur Steuergeräte- und Busanalyse

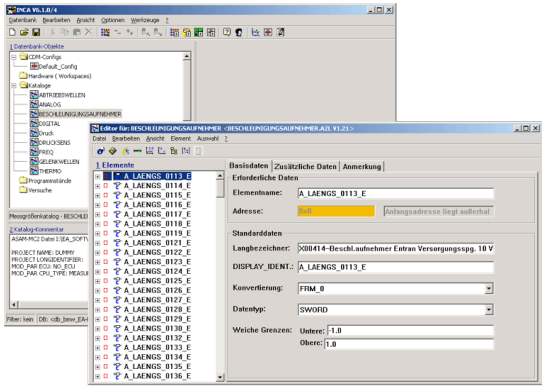
- verschiedenste in Entwicklung und Absicherung verwendet (CARMEN TopEd, Logfilekonverter, MoniCar, CANoe, INCA)
- Lösungen für spezielle Anwendungsfälle müssen mehrmals implementiert werden



Logfilekonverter



CANoe

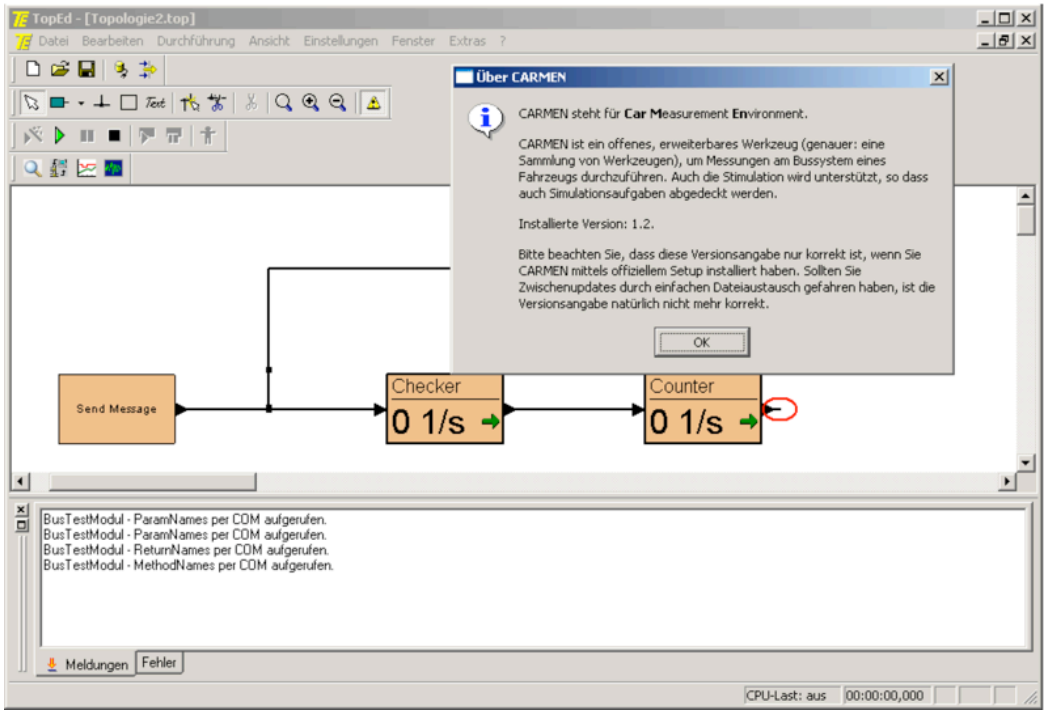


INCA

CARMEN (Car Measurement Environment)

- Eigenentwicklung der BMW Group
- Sammlung von Werkzeugen zur Messung an Bussystemen (CAN, FlexRay, LIN und MOST)
- Simulation ebenfalls unterstützt
- offline Analyse: Logfilekonverter
- online Analyse: TopEd (Topologie Editor)

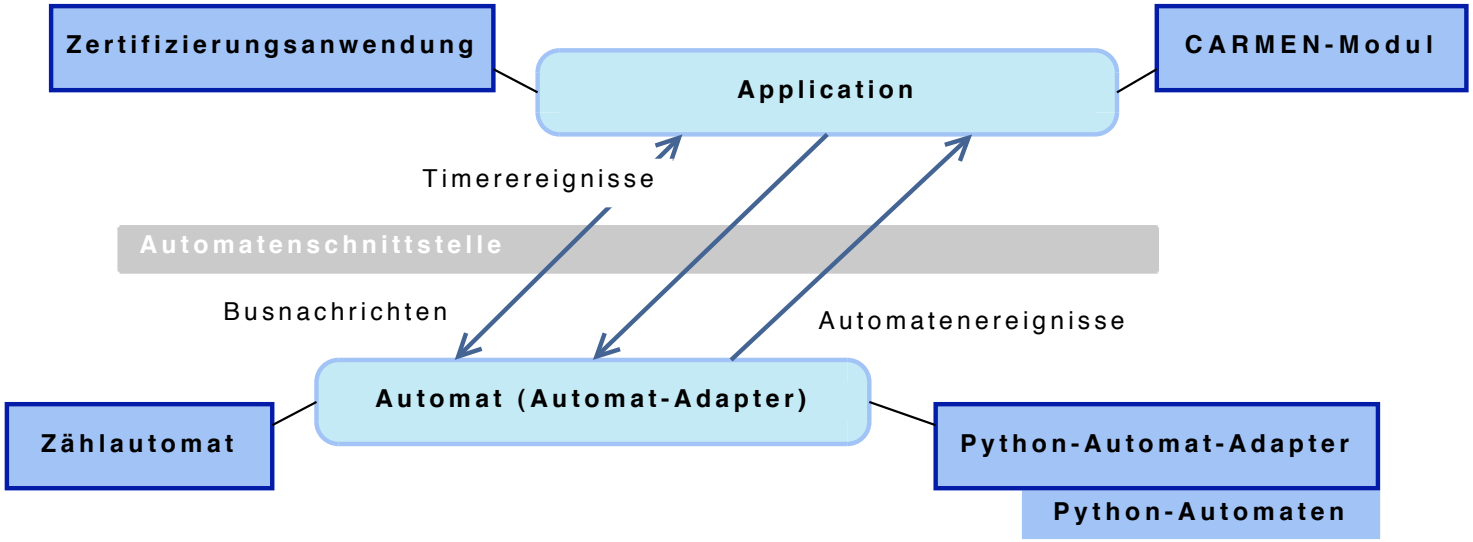
CARMEN TopEd



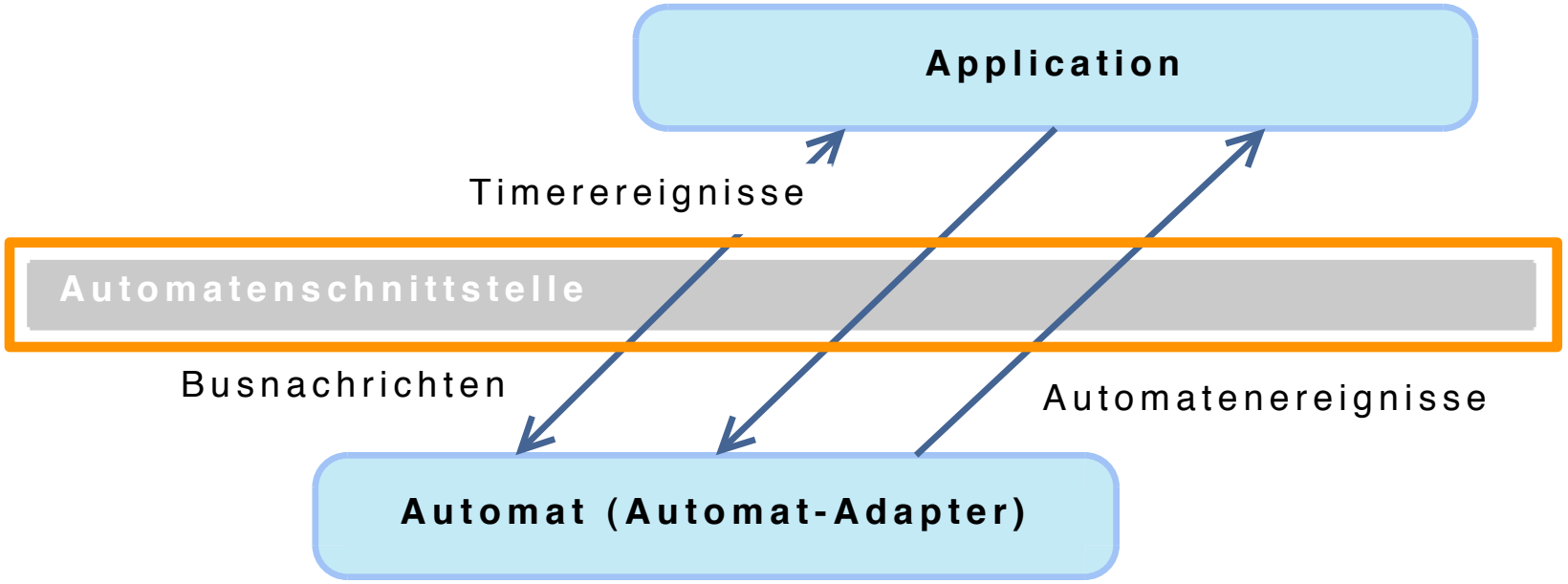
- Quell- und Analysemodule als Blöcke dargestellt
- entsprechend der gewünschten Funktion vernetzt
- Erweiterung durch eigene Module möglich

Ziel der Arbeit

- Schaffung einer einheitlichen Schnittstelle zum Anbinden von Analysebausteinen (Automat oder Automat-Adapter) an verschiedene Analysewerkzeuge



Automatenschnittstelle

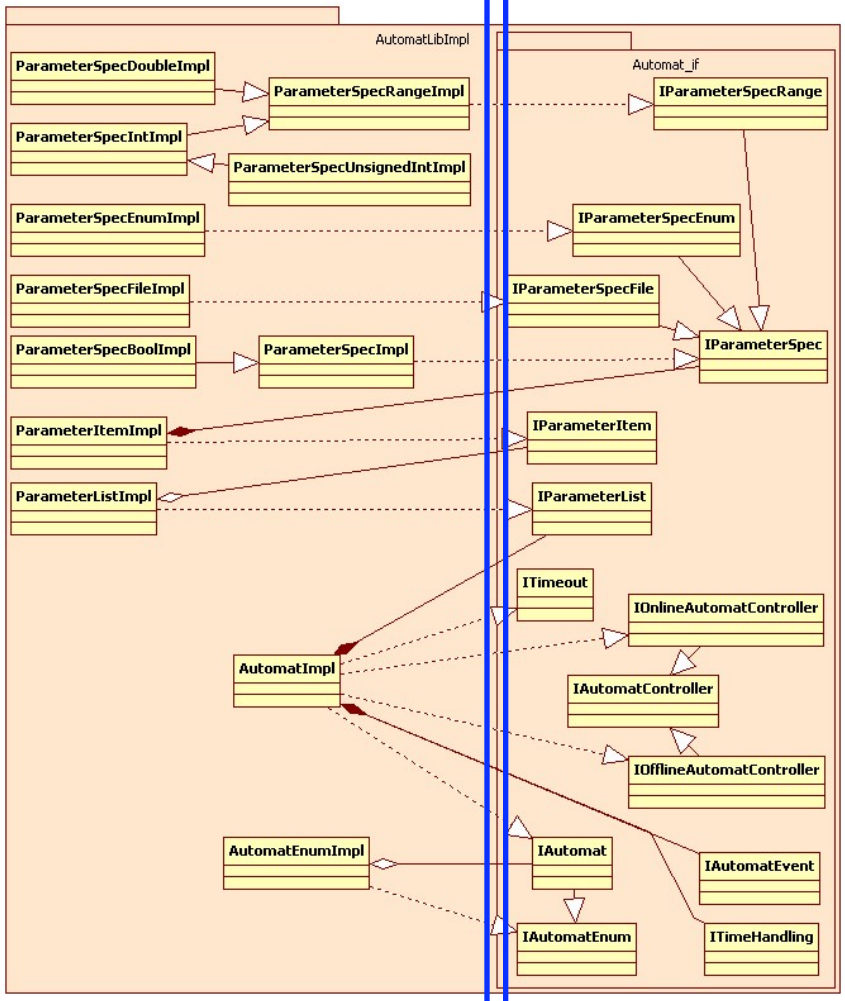


Merkmale

- orientiert sich an der API von CARMEN (C++)
- Rohdaten der Busnachrichten übertragen
- für online-Automaten Timerereignisse und Abfrage der Testzeit möglich
- für offline-Automaten keine Anforderung an die minimale Auswertegeschwindigkeit
- Anbinden von Automaten oder Adaptern
- Rückgabe von Busnachrichten, Tracemeldungen und Fehlerereignissen (OnError, OnTimeout)

Automatenschnittstelle

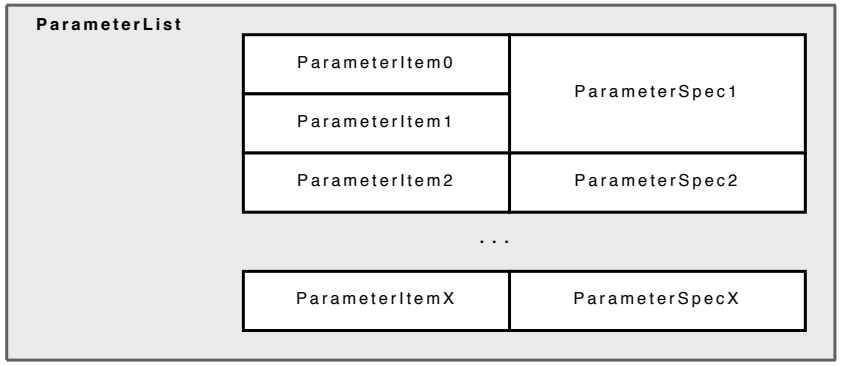
Prototypen



Interfaces

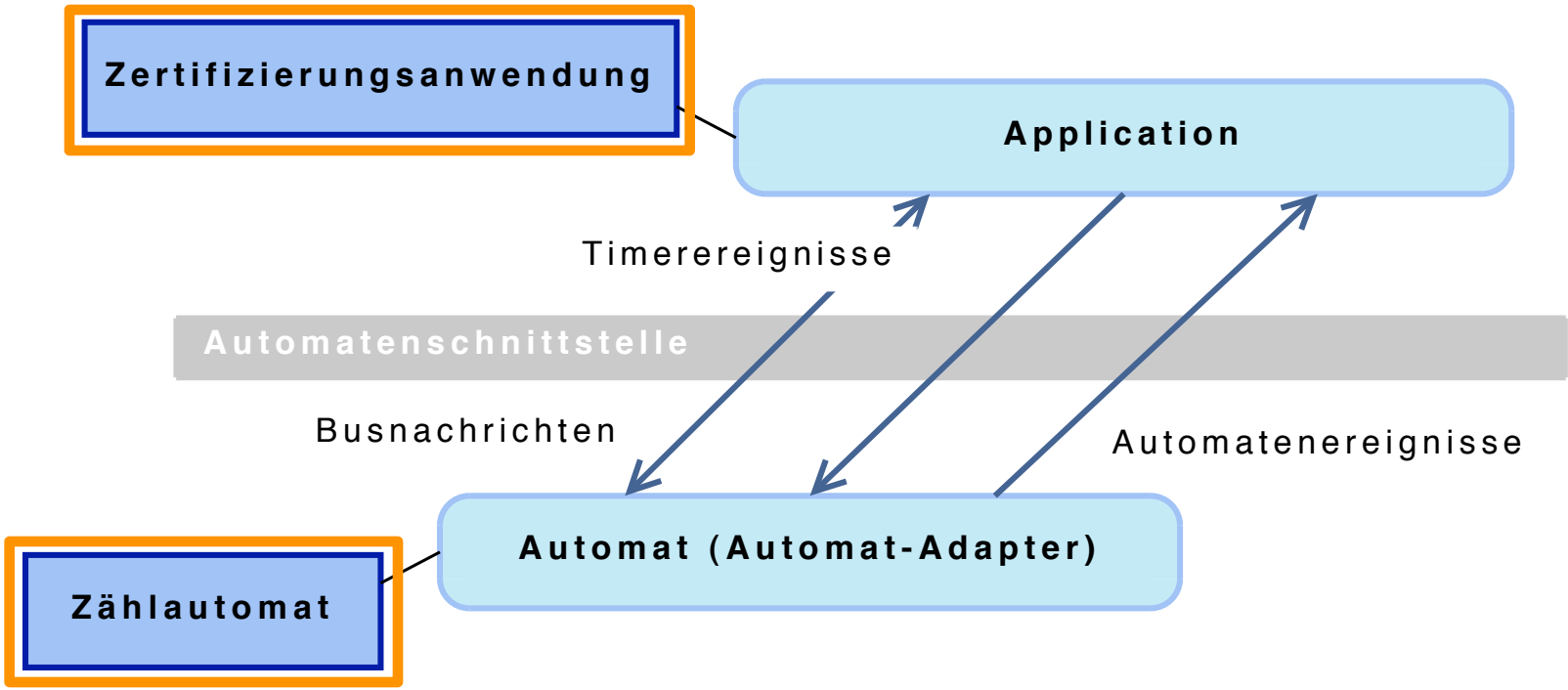
Implementierungen

- Prototypen der Schnittstelleninterfaces
  - AutomatEnum: Auflisten aller Automaten
  - ParameterList, ParameterItem, ParameterSpec: dynamische Handhabung von Parametertypen
  - Automat



- grundsätzlich nötige Dialoge
  - Dialog zur Auswahl des Automaten
  - generischer Dialog zum Editieren der Parameter

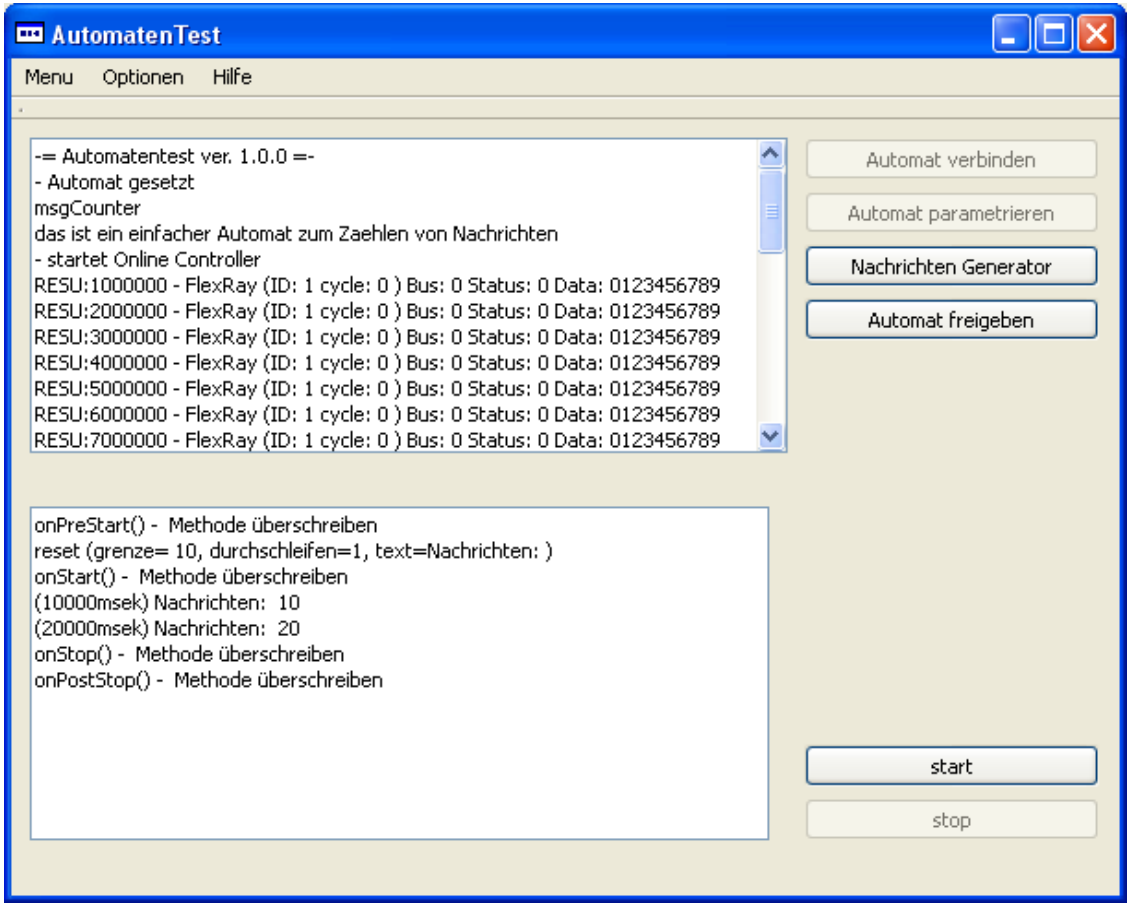
Zertifizierungsanwendung und Zählautomat



Zertifizierungsanwendung

- Überprüfung von beliebigen Automaten (auch von Drittanbietern)
- zum Abbilden verschiedenster Anwendungsfälle
- online- und offline-Betrieb möglich
- beliebige Nachrichten für CAN, FlexRay etc. erzeugen
- intuitiv bedienbare moderne Benutzerschnittstelle
- Ausgabe der gesendeten und empfangenen Nachrichten, sowie der Meldungen vom Automaten

Zertifizierungsanwendung



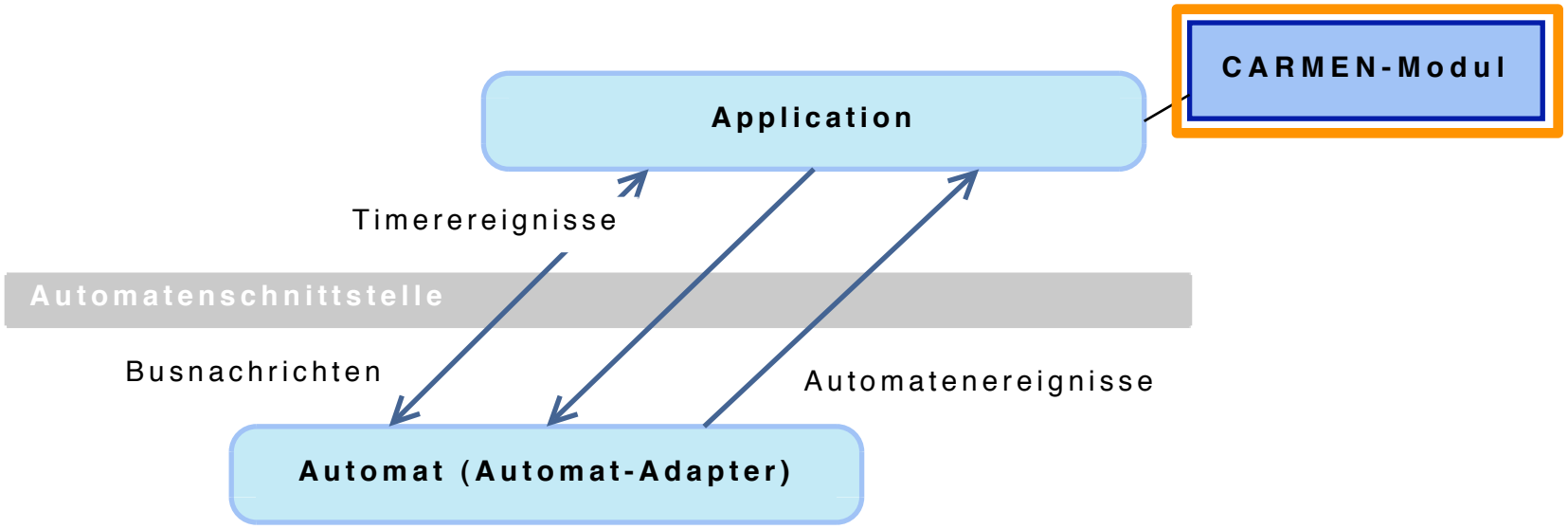
Schwierigkeiten

- Äquivalente Simulation von offline- und online-Betrieb
- Generatoren in Warteschlange (Hash-Map)
- Hash-Wert bezeichnet Zeitpunkt des nächsten Ereignisses
- nach Nachrichtengenerierung wird nächster Zeitpunkt ermittelt und wieder der Warteschlange zugefügt
- Threading durch QTimer realisiert
- offline-Mode: Verzögerung von 0 - wartet nur auf GUI-Aktualisierung
- online-Mode: Verzögerung berechnet

Zählautomat

- einfacher Automat zum Zählen von Nachrichten
- aktueller Stand des Zählers wird in regelmäßigen Abständen als Tracemeldung ausgegeben
- Intervall, Text der Tracemeldung, Weitergabe der Nachrichten an Ausgang parametrierbar
- zum Testen eigener Implementierungen der Schnittstelle
- Beispiel zur schnellen Einarbeitung

CARMEN-Modul



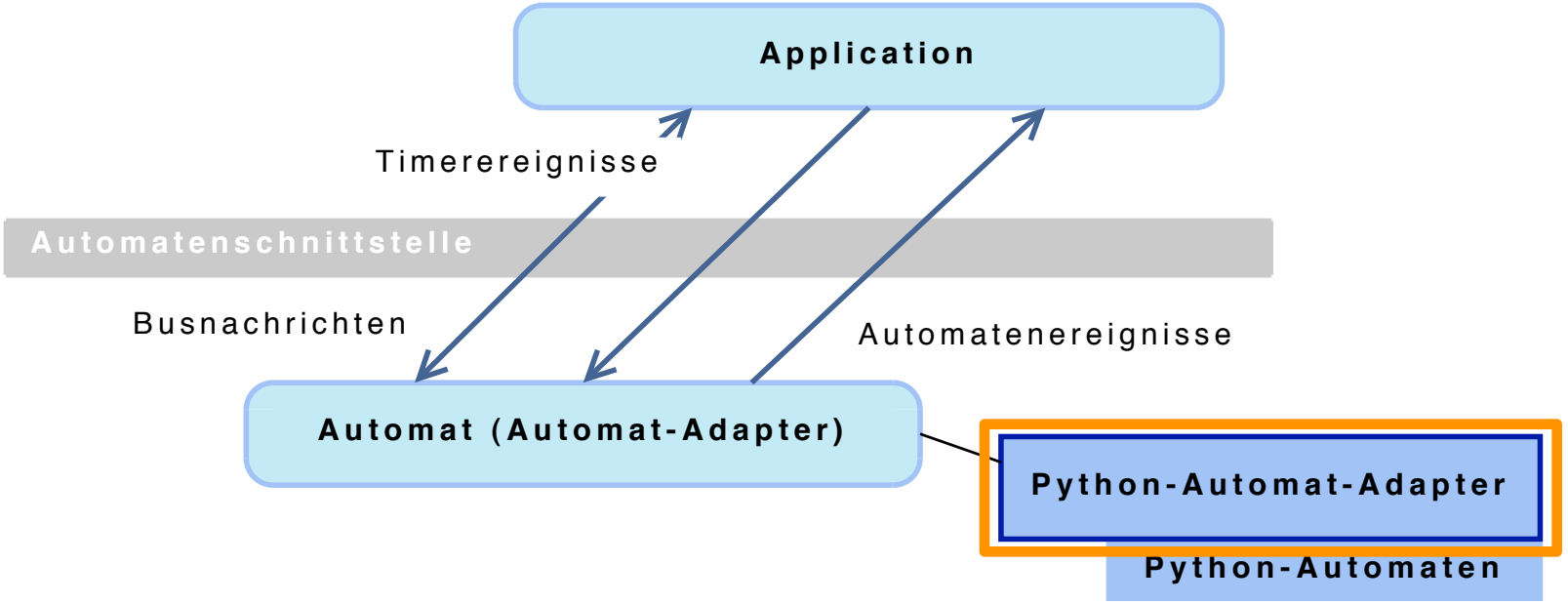
Merkmale

- für CARMEN-TopEd
- beliebig oft in Topologie einfügbar
- unterstützt Module, die für den online-Modus erstellt wurden
- Laden und Entladen des Automaten möglich
- Tracemeldungen und Fehlernachrichten werden in der Textkonsole von TopEd ausgegeben

Äquivalente Implementierungen

- CANalyzer/CANoe und INCA
  - kein Zugang zum Quellcode
  - nur Realisierung über Automatisierungsschnittstelle möglich
  - sehr eingeschränkt und aufwendig
- Logfilekonverter
  - möglich, aber noch nicht realisiert

Python-Automat-Adapter



Python

- plattformunabhängige, objektorientierte und dynamische Programmiersprache
- wird zur Laufzeit interpretiert
- zur Geschwindigkeitsverbesserung wird ein Compilat erzeugt

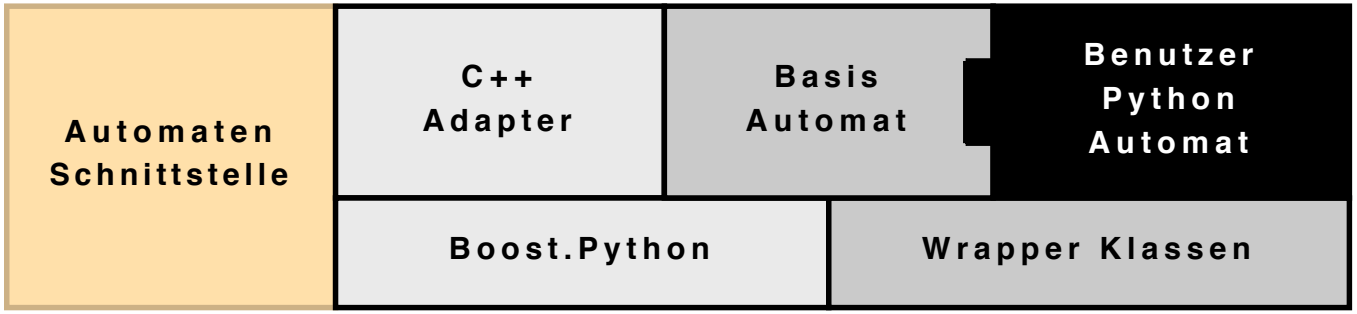
Warum?

- rapid Prototyping
- spezielle Anwendungfälle realisierbar

Problem

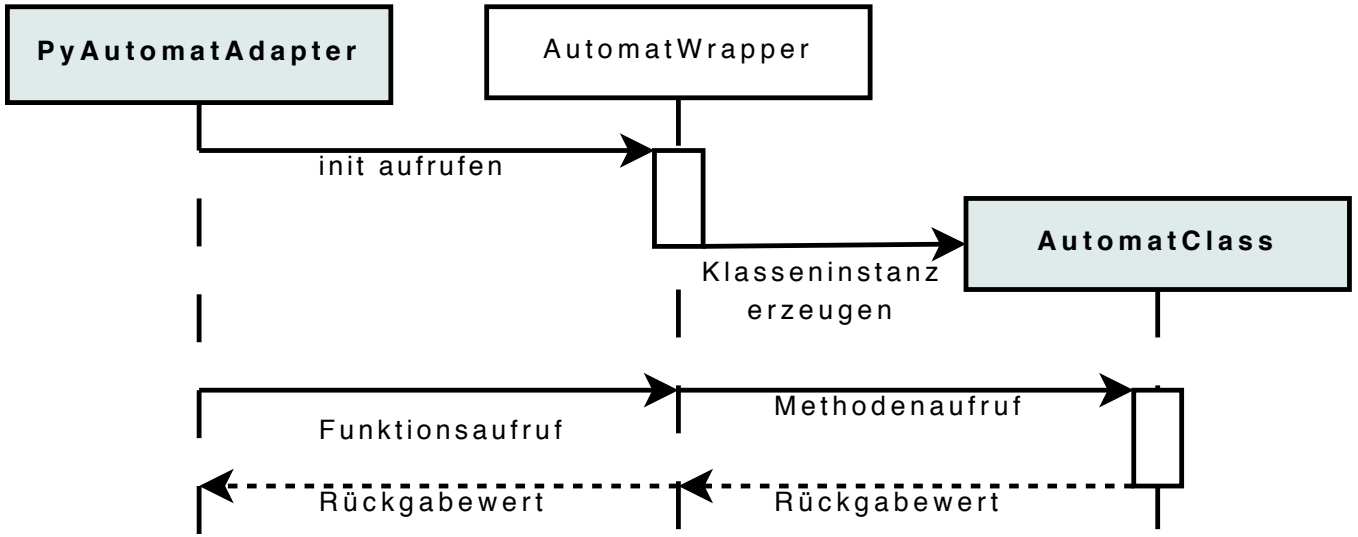
- Schnittstelle in C++ realisiert
- Einbetten (embedding) von Python nötig
- verschiedene Bibliotheken zur Vereinfachung vorhanden (Elmer, Python C/API, SWIG, Boost.Python)
- Boost.Python gewählt da:
  - embedding gut unterstützt und dokumentiert
  - für viele Betriebssysteme verfügbar
  - Bibliotheken in zahlreichen Projekten bereits verwendet und aktive Weiterentwicklung

Aufbau



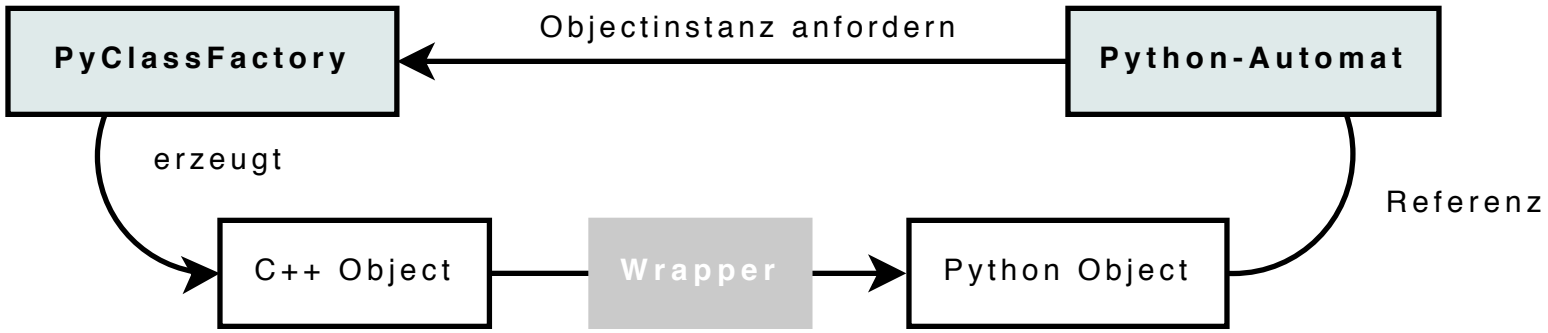
- grundlegende Funktionsweise in Python wiedergeben
- Änderung der Handhabung von Timerereignissen
- Nachrichten und Nachrichtenbytes als Datenklassen übergeben
- Basisautomatenklasse zum Ableiten eigener Automaten
- Beispielautomat zur leichteren Einarbeitung

Schwierigkeiten



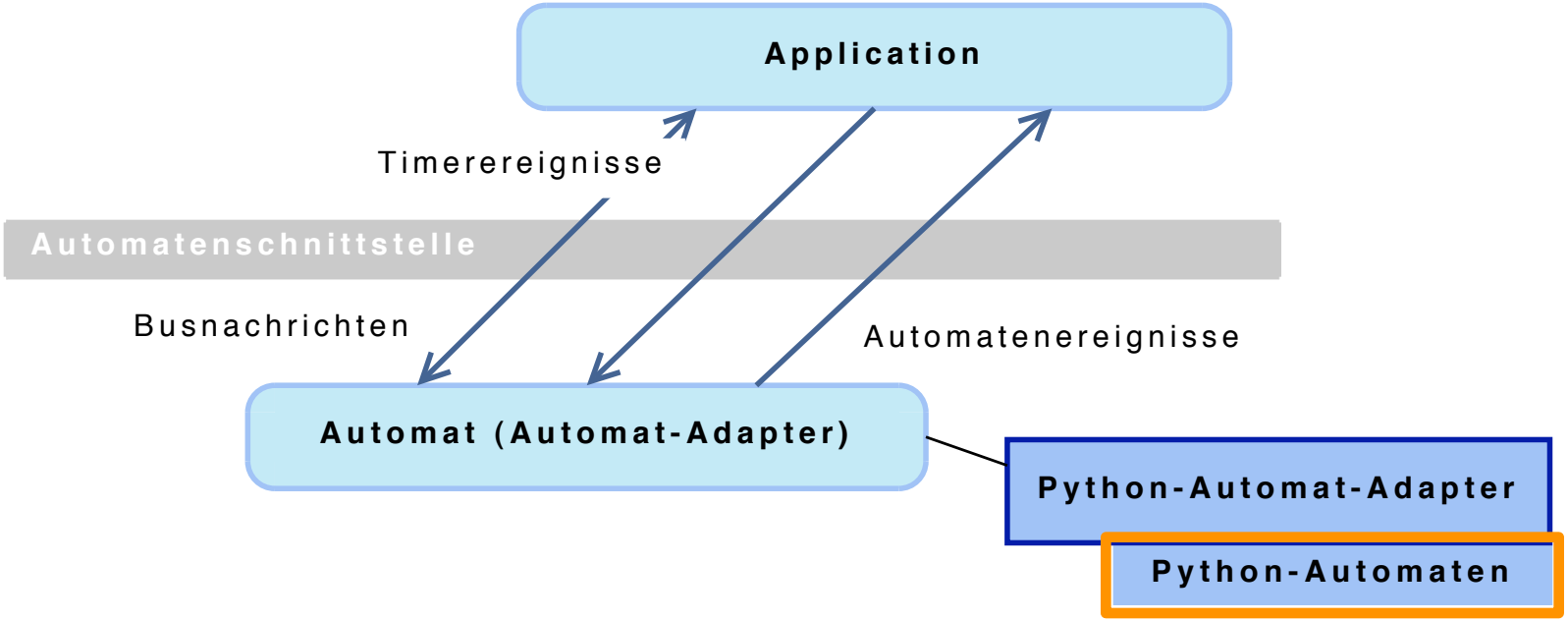
1. keine direkte Instantiierung der Python-Automatenklasse möglich
   - über Phython-Bibliothek gewrappt
   - Automatenklasse wird in Namensraum geladen
   - Funktionen rufen Klassenmethoden auf

Schwierigkeiten



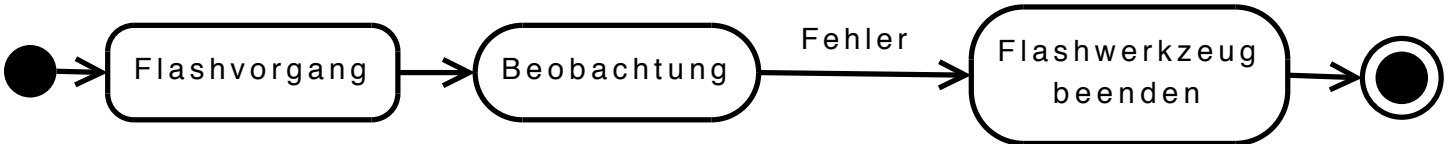
2. Erzeugung von Python-Klassen, die wieder nach C++ importiert werden sollen problematisch
   - Klassen-Factory in C++ realisiert
   - Referenz zur Factory bei Automatenerzeugung übergeben
   - Instanzen von Factory angefordert und nach Python gewrappt

Anwendungsfälle



Beobachtung des Flashprozesses

- Flashen für verschiedene Kombinationen aus Programm- und Datenstand automatisiert testen



- Beobachten des Flashens über den FlexRay-Bus
- bei Eintreten eines Fehlerfalls, beenden des Flashprogrammes

Zusammenfassung

- Schnittstellendefinition überprüft und korrigiert
- Prototypen für die Elemente der Schnittstelle erstellt
- Anwendung der Automatenschnittstelle realisiert
  - Zertifizierungsanwendung und Zählautomat
  - Modul für CARMEN TopEd
  - Python-Automat-Adapter und zahlreiche Python- Automaten
- durch Anwendungsfälle von aktuellen Problemen aus der Absicherung praktische Relevanz dargestellt

Ausblick

- Interpretation der Rohdaten
- weitere Werkzeuge an Schnittstelle anbinden (zum Beispiel Logfilekonverter)
- weitere Automaten realisieren (zum Beispiel für Projekt “SMARD”, Matlab)

Vielen Dank

für Ihre Aufmerksamkeit

Bibliotheken zum Verbinden von Python und C/C++

Bibliothek	Nachteile	Vorteile
Python/C API	• direkter Zugang, keine zusätzlichen Bibliotheken nötig	• Klassen müssen in C++ und Python erstellt werden • Referenz Zähler muss selbst aktualisiert werden
Elmer	• sehr schnelle einfache Implementierung	• Visual C++ Compiler nicht unterstützt • nicht optimal für komplexe Projekte
SWIG	• schnelle Wrappergenerierung für komplexe Klassenstrukturen • für eine Vielzahl von Sprachen verfügbar	• Schwerpunkt auf extending, embedding sehr gering betrachtet
Boost.Python	• Framework für viele Projekte verwendet • Zusatzwerkzeuge zum Generieren der Wrapperklassen verfügbar	• nur einzelner Thread möglich

spezifische Automaten

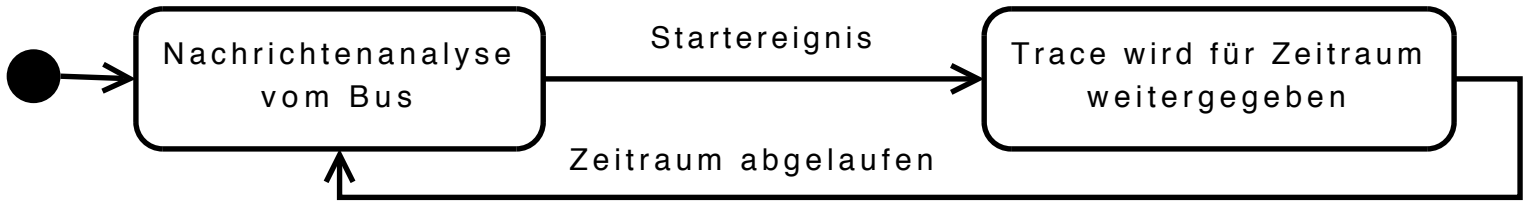
- Beobachtung des Flashprozesses
- Vorfiltern von Nachrichtentraces
- Überwachung der Liste aller BOS-Dienste

Verknüpfung zur Trace-Analyse von ECU-TEST

- Überwachung der EWS-Freischaltung

Vorfiltern von Nachrichtentraces

- oftmals keine Traces über kompletten Zeitraum benötigt



- Nachrichtenfluss nicht an Ausgang weitergeben
- beim Eintreffen des Start-Ereignisses (Fehler, Problem, markante Nachricht) - Beginn des Weiterleitens der Nachrichten
- Stopp nach einem festen Zeitraum

Benutzung der Zertifizierungsanwendung

- Verbinden zu beliebigen Automaten
- Parametrierung des Automaten
- Auswahl der gewünschten Nachrichtengeneratoren (besteht aus Zeitbedingung und Nachrichtentyp)
- Teststart im offline- oder online-Betrieb
- Ausgabe der Tracemeldungen, Ergebnisnachrichten und Fehlerereignisse

Automatenschnittstelle-Interfaces