Introduction to different Interfaces, Technologies and APIs

CANoe as Open Environment
Agenda

1. Current Tool Landscape
2. What CANoe Can Do
3. Interfaces
4. Conclusion
Current Tool Landscape

Motivation
Current Tool Landscape

Motivation

- Many different specialized tools involved – one for each domain
- Tools have to interact with each other
  - Often done with homegrown solutions
    - Drawback
      - Large effort to maintain compatibility over several versions
      - Project specific – low reuse

→ Thus using already available interfaces is obvious
1. Current Tool Landscape
2. What CANoe Can Do
3. Interfaces
4. Conclusion
Most Common modeling tool is Matlab/Simulink
- CANoe provides a Matlab/Simulink Interface
- More generic approach: FMI (Functional-Mockup-Interface)
- CANoe can load and integrate FMUs
- Vector specific: FDX
  - UDP based and thus commonly available
What CANoe Can Do

Automated Testing

- ASAM XiL API
  - Standardized interface
- COM
  - Standardized interface from Microsoft
  - Only suitable to setup/control test
- Vector specific: FDX
  - UDP based and thus commonly available
  - Also for high performance data exchange
- National Instruments
  - LabVIEW - Shared Network Variables

Tool 1 HIL/RBS

Tool 5 Autom. Test

SUT

CAN, LIN, FlexRay, Most, Ethernet, ...
1. Current Tool Landscape
2. What CANoe Can Do
3. Interfaces
4. Conclusion
Matlab/Simulink Interface

Application Model in MATLAB

Application Model in CAPL

on timer tMoveDownTimer
{
    if(gPosWN < 15 && gMoveDown == gTrue)
    {
        gPosWN++;
        $DOOR_l::WN_Position = gPosWN;
    }
}

IL: Interaction Layer
NM: Network Management
TP: Transport Protocol
Matlab/Simulink Interface

- Provide the environment for a SUT in a compiled MATLAB model
- Stimulation of sensors/actuators from model outputs/inputs
- Utilizing e.g. CANoe RT for deterministic timing behavior

![Diagram showing the Matlab/Simulink Interface with interfaces for SUT, CANoe, Signal Input/Output, Sensor Input/Output, SW Stimulation/Reaction, and Remaining Bus and Application/Environment/Vehicle Model.]
**FMI (Functional Mockup Interface)**

- Idea: Define an interface for integrating models of different types (mechanical, electrical, ...) and vendors and make them available to different simulation platforms without disclosing IP.

![Functional Mockup Interface for model exchange and tool coupling](image)

- FMI for model exchange
- FMI for co-simulation
Interfaces

FMU Export
FDX

- UDP based link – therefore generally available
- FDX (Fast Data eXchange) is a protocol for simple, fast, real-time exchange of data between CANoe and other systems via an Ethernet connection. The protocol enables other systems read and write access to CANoe system variables, environment variables and contents of bus messages.

FDX protocol is easy to use
- UDP is available on almost every system
- Customer can implement specific solutions on his own
The type of data can be any typical integer or floating point type, scalar or array and even string.

Data elements are sent and received in groups.
Interfaces

FDX Editor

- Data groups
- Configured Symbols in a data group
- Symbol Selection
- HIL Symbols (optional)
- Layout view
Interfaces

FDX – Configuration in CANoe with XML

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<canoe_fdxdescription version="1.0">
  <datagroup groupID="1" size="8">
    <identifier>EasyDataRead</identifier>
    <item type="int16" size="2" offset="0">
      <identifier>SigEngineSpeed</identifier>
      <signal name="EngineSpeed" value="raw"/>
    </item>
    ...
  </datagroup>

  <datagroup groupID="2" size="8">
    <identifier>EasyDataWrite</identifier>
    <item type="int16" size="2" offset="0">
      <identifier>EnvEngineSpeedEntry</identifier>
      <sysvar name="EngineSpeedEntry" namespace="FDX"/>
    </item>
    ...
  </datagroup>
</canoe_fdxdescription>
```
FDX – Implementation on Client Side with any Programming Language

```c
int main(int argc, char* argv[]) {
    const char* addr = "127.0.0.1";
    if (argc == 2) {
        addr = argv[1];
    }
    gFDXSocket.SetCANNowAddr(addr, 2009);
    gFDXSocket.Open();

    const int uCycleInterval = 100;

    int currentEngSpeed = 0;
    int deltaEngineSpeed = +50;

    // build datagram
    gFDX Datagram.InitRichHeader();
    gFDX Datagram.AddDataRequest(EasyDataRead::roGroupID);
    void* dataBytes;
    dataBytes = gFDX Datagram.AddDataExchange(EasyDataWrite::cGroupID, EasyDataWrite::cSize);
    EasyDataWrite* writeData = reinterpret_cast<EasyDataWrite*>(dataBytes);
    writeData->VarDataEntry = currentEngSpeed;
    writeData->VarDataStateSwitch = (gCANetMeasurementState==CANNetFDX::kMeasurementState_Running)?1:0;
    writeData->SigFlashLight = gHeadLightsState;
    writeData->SigHeadLight = gHeadLightsState;

    // send datagram
    gFDXSocket.Send(gFDX Datagram);

    // receive and dispatch data from CANoe
    int i = 0;
    if (gFDXSocket::OK) {
        gFDXDispatcher.DispatchDatagram(gFDX Datagram);
    }
}
```
Interfaces

ASAM XiL API

Test Automation Tool

XIL Framework (optional)

Mapping

XIL Ports

Model Access Port  Network Port  Diag Port  EES Port  ECUM Port  ECUC Port

Test Bench

HIL Hardware / Offline Simulation
ASAM XiL API – Usage scenarios

Scenario 1: CANoe as Server

3rd Party Test Automation Tool

Test Cases

XIL API

CANoe

Scenario 2: CANoe as Client

vTESTstudio

Test Cases

CANoe

XIL API

3rd Party Test Bench
ASAM XiL API – CANoe as Server

Scenario 1:
CANoe as Server

3rd Party Test Automation Tool

Test Cases

XIL API

CANoe

Scenario 2:
CANoe as Client

vTESTstudio

Test Cases

CANoe

XIL API

3rd Party Test Bench
Interfaces

XIL API – CANoe as Server
ASAM XiL API – Usage scenarios

Interfaces

Scenario 1: CANoe as Server

- 3rd Party Test Automation Tool
  - Test Cases

- XIL API

- CANoe

Scenario 2: CANoe as Client

- vTESTstudio
  - Test Cases

- CANoe

- XIL API

- 3rd Party Test Bench
Interfaces

XIL API – CANoe as Client
Interfaces

ASAM XiL API – Supported ports

Model Access Port

Variable Access
Generators
Capturing

EES Port

Short Circuit Errors
 Interruption Errors
 Dynamic Timing
 Static Timing
 Pin to Pin Errors
 Resistor Errors
 Loose Contact Timing

Network Port

Diag Port

ECUM Port

ECUC Port

Largely supported
Partly supported
Not supported
LabVIEW Shared Network Variables

- Interface to LabVIEW
  - Realization: Usage of the Shared Network variable API of LabVIEW

- Measurement signal configuration in CANoe via GUI
  - Fully automated generation of required system variables

Diagram:

- GUI PC
  - CANoe GUI
  - Configuration

- RT PC
  - CANoe RT
  - Data exchange

- LabWindows CVI Runtime
- NI Real Time Rack
- LabVIEW VI
LabVIEW Shared Variables
COM

- CANoe provides a COM server with numerous object to
  - Remote control CANoe, e.g. start and stop the measurement
  - Build up/modify configuration dynamically
- COM server is not suitable for exchanging simulation data in an online setup
Microsoft COM
1. Current Tool Landscape
2. What CANoe Can Do
3. Interfaces
4. Conclusion
Conclusion

- The “oe” in CANoe means “Open Environment”
- Many available interfaces are already supported
Available Interfaces

Tool 2
Env. Mdl.

Tool 3
Dynamic Mdl.

Tool 4
ADAS, virt. driving

Tool 5
Autom. Test

Matlab Interface, FMI

FMI, FDX

ASAM XIL API, FDX, COM

AIO/DIO

CAN, LIN, FlexRay, Most, Ethernet, ...

SUT
Open Environment by APIs

Test Management
- Requirements
- Test Cases
- Test Planning
- Test Result Analysis

Additional Models
- Matlab
- IPG CarMaker
- TESIS
- TESIS
- Other 3rd party tools

Vector Tools
- Test Design
  - vTESTstudio
- Simulation Model
  - CANoe

Vector Network Interfaces
- Direct Integration
- Shared Network Variables
- ASAM XIL API
- FDX
- .NET DLL

VT System
- LabVIEW
- XIL API
- Other 3rd-party HIL System
- ...

SUT
- NI
- Matlab/Simulink Integration
- FMI/FMU
- DLL
- FDX
- .NET DLL

Traceability Matrix
- XML Report

Requirements
- Trace Items
- ReqIF
- ReqIF Exp.
For more information about Vector and our products please visit

www.vector.com

Author:
Neuffer, Jochen
Vector Germany