FRANCA to Adaptive AUTOSAR
Compatibility study between Franca IDL + CommonAPI and AUTOSAR ARA::COM

Marc BELLANGER, Renault Software Labs
Klaus BIRKEN, Itemis AG
Christopher SCHWAGER, ITK Engineering GmbH
Torsten MOSIS, Systemticks GmbH

Presentation to AMM, Munich, 15 May 2019
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:00 - 11:15</td>
<td>Motivation</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Objectives</td>
</tr>
<tr>
<td>11:15 - 11:40</td>
<td>Model level</td>
<td>Mapping of models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transformation tooling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q&amp;A / discussion</td>
</tr>
<tr>
<td>11:40 - 11:55</td>
<td>Runtime</td>
<td>Demonstrator - from CES to AMM</td>
</tr>
<tr>
<td>11:55 - 12:15</td>
<td></td>
<td>Concept mapping issues and mitigation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q&amp;A / discussion</td>
</tr>
<tr>
<td>12:15 - 12:20</td>
<td></td>
<td>Logging &amp; Tracing</td>
</tr>
<tr>
<td>12:20 - 12:30</td>
<td>Next steps</td>
<td></td>
</tr>
</tbody>
</table>
INTRODUCTION
Marc BELLANGER, Renault Software Labs
Introduction

SOA (service oriented architecture) is a clear SW design trend in automotive. SOA in automotive can be represented in 3 layers:

- **Interface Description Language (IDL)**: Define the contract between services and clients.
- **Middleware**: Generates code (from IDL) and libraries used by SOA services and clients to communicate. Abstracts the underlying transport protocol.
- **Transport protocol**: Provide rules to transport the messages (message identification, serialization, ...).
Problem statement

GENIVI study demonstrated that 2 SOA technologies are under the automotive spotlights today.

- COMMONAPI + FIDL: strongly adopted in infotainment domain
- ARA::COM: Autosar emerging technology

⇒ How to ensure compatibility?
OBJECTIVES
Marc BELLANGER, Software Architect, Renault Software Labs
Objectives

To ensure compatibility we have to reach 2 objectives

1. Talk the **same language**/Use the **same** communication concepts
   a. Choose one IDL as reference and translate it
      i. FRANCA IDL chosen as reference for readability (ARXML is XML format)
      ii. Goal is the 2 ways translation
   b. IDL brings a set of communication concepts:
      i. Message types: Event, RPC calls, …
      ii. Data types: unitary data (UInt8, Float, …) and composed data (Struct, …)

2. Transport the information in the **same format**
   a. Uniquely identify the messages
   b. Serialize the data in the same order.
      ⇒ SOME/IP is the solution to align transport format.
Example of SOA workflow
Model-level Mapping of AUTOSAR Adaptive and Franca IDL

Dr. Klaus Birken, itemis AG
Model transformation tooling

Major goal: Transform models such that the resulting code on both sides will be compatible wrt. its IPC properties.
Definition of mapping

- AUTOSAR metamodel defined by Artop metamodel (artop.org)
- Franca IDL metamodel defined by Franca Eclipse project
- mapping between both domains is defined
  - on a conceptual level (e.g., AUTOSAR service ⇔ Franca interface)
  - on a metamodel level (e.g., ClientServerOperation ⇔ FMethod)
- the metamodel level mapping is the starting point for tool implementation
- mapping table (GoogleDocs format)
### Definition of Mapping Franca/AUTOSAR

This is the specification for the transformation AUTOSAR Adaptive <-> Franca. We disregard all concepts of AUTOSAR which purely belong to AUTOSAR Classic.

#### Mapping table

| Group | Franca concept or one of (IGNORE | ERROR | EMULATE) (see A10) | Franca metamodel export ID | Franca metamodel classifier | Franca metamodel attribute | AUTOSAR concept or one of (IGNORE | ERROR | EMULATE) (see A10) | Artop metamodel classifier | Artop metamodel attribute | Detail level (IDL, Serialization, CommonAPI, SOME/IP) |
|-------|-------------------------------------------------|----------------------------|----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|----------------------------------|
| structure | version of type collection | IDL1190 | Class FTypeCollection | FVersion version (optional) | see IDL1490 | see IDL1490 | see IDL1490 | IDL |
| structure | list of types (all with visibility public) | IDL1200 | Class FTypeCollection | List<FType> types | package contents | ARPackage | List<PackageElement> getElements() | IDL |
| structure | list of constants (all with visibility public) | IDL1210 | Class FTypeCollection | List<FConstantDef > constants | asked MBR... | | | IDL |
| structure | interface definition | IDL1220 | Class FInterface | n/a | interface definition | ServiceInterface | | IDL |
| structure | list of attributes | IDL1230 | Class FInterface | List<FAttribute> attributes | fields of a service | ServiceInterface | List<FField> getField() | IDL |
| structure | list of methods | IDL1240 | Class FInterface | List<FMethod> methods | client server operations of a service | ServiceInterface | List<ClientServerOperation> on the service interface | IDL |
| structure | list of broadcasts | IDL1250 | Class FInterface | List<FBroadcast> broadcasts | events of a service | ServiceInterface | List<VariableDataPrototype> getEvents() | IDL |
| structure | optional interface contract | IDL1260 | Class FInterface | FContract contract (optional) | IGNORE | n/a | n/a | n/a |
| structure | inheritance for interfaces | IDL1270 | Class FInterface | FInterface base (optional) | EMULATE | n/a | n/a | IDL |
| structure | manages-relation for interfaces | IDL1280 | Class FInterface | List<FInterface> managed Interfaces | ERROR | n/a | n/a | IDL |
| comm primitives | method | IDL1290 | Class FMethod | n/a | operation | ClientServerOperation | n/a | IDL |
| comm primitives | fire-and-forget flag | IDL1300 | Class FMethod | EBoolean fireAndForget (optional) | fire-and-forget flag | ClientServerOperation | boolean isSetFireAndForget() | IDL |

* read-only link to document on GoogleDocs: [mapping table](https://docs.google.com/document)
How to resolve mapping problems?

● objective: generated code is compatible

● reasons for incompatibilities:
  a. no corresponding concept on metamodel level (e.g., inheritance)
  b. generated code shows different behavior (e.g., error handling)

● options for resolving incompatibilities:
  a. check if concept can be “emulated” (e.g., flattening inheritance)
  b. check if code generation can be fixed (either by adapting the code generator or indirectly by changing the mapping)
  c. if all else fails: make user aware that concept cannot be mapped (e.g., by providing specific validation checks)
Transformation Tooling: Current Status, Usage and Roadmap

Dr. Klaus Birken, itemis AG
Model transformation tooling: Status

- prototype is available ("stage 1"), part of work for CES 2019 demonstrator
  - bi-directional, but supports only limited subset of mappings
  - transformations can be used in the Eclipse IDE only
  - no automatic build, only limited test cases
  - based on AUTOSAR Adaptive Platform R18-03
- development of near-production ready tool has been approved by GENIVI beginning of May 2019
- public repository: [https://github.com/GENIVI/franca_ara_tools](https://github.com/GENIVI/franca_ara_tools)
Model transformation tooling: Usage

- installation in Eclipse as described in README (see github repo)
- currently, transformation is executed only programmatically (e.g., via JUnit test)

```
// load example Franca IDL interface
val inputfile = "input.fidl"
val FModel fmodel = loader.loadModel(inputfile)
assertNotNull(fmodel)

// transform to arxml
val conn = new ARACConnector
val fromFranca = conn.fromFranca(fmodel) as ARAModelContainer
conn.saveModel(fromFranca, "result.arxml")
```
Model transformation tooling: Roadmap

- project start stage 2: started May 2019
- tasks
  - actual transformations (both directions)
  - test cases based on simple and real-world models
  - analysis of SOME/IP deployment mapping
  - automatic build of the tools
  - installable features, command-line tool
- releases for beta testers: continuously (CI build)
- release 1.0 (planned): October 2019
Demonstrator – from CES to AMM

Christopher Schwager, Senior Expert Embedded Architectures, ITK Engineering GmbH
Demonstrator – from CES to AMM

Does the transformation tool provide valid results?

Interoperability on target level?

Could the GENIVI Development Platform and an Adaptive AUTOSAR ECU perform joint functionality?
Emergency Brake Assistant Scenario
Steps Executed to Establish the Communication

1. Interface Definition

2. Model-to-Model Transformation

3. SOME/IP Deployments

4. Generation of the proxies and skeletons

5. Integration of the generated code
package genivi.aasr.showcase

interface IDrivingLane {
    ...
    struct LaneType {
        UInt16 frameId
        UInt32 intersectionPointX
        UInt32 intersectionPointY
        UInt32 lowerRightPointX
        UInt32 lowerRightPointY
        UInt32 lowerLeftPointX
        UInt32 lowerLeftPointY
    }

    broadcast LaneDetected {
        out {
            LaneType drivingLane
        }
    }
}
package genivi.aasr.showcase

interface IDrivingLane {
  ...
  struct LaneType {
    UInt16 frameId
    UInt32 intersectionPointX
    UInt32 intersectionPointY
    UInt32 lowerRightPointX
    UInt32 lowerRightPointY
    UInt32 lowerLeftPointX
    UInt32 lowerLeftPointY
  }

  broadcast LaneDetected { out { LaneType drivingLane } }
}

* ARXML is shortened for the presentation*
package genivi.aasr.showcase

interface IDrivingLane {
    ...
    struct LaneType {
        UInt16 frameId
        UInt32 intersectionPointX
        UInt32 intersectionPointY
        UInt32 lowerRightPointX
        UInt32 lowerRightPointY
        UInt32 lowerLeftPointX
        UInt32 lowerLeftPointY
    }
    broadcast LaneDetected { 
        out {
            LaneType drivingLane
        }
    }
}

<Service-Interface>
<Short-Name>IDrivingLane</Short-Name>
<NAMESPACES>
  <SYMBOL-PROPS>
    <Short-Name>genivi</Short-Name>
    <Symbol>genivi</Symbol>
  </SYMBOL-PROPS>
  <SYMBOL-PROPS>
    <Short-Name>aasr</Short-Name>
    <Symbol>aasr</Symbol>
  </SYMBOL-PROPS>
  ...
</NAMESPACES>
<EVENTS>
  <VARIABLE-DATA-PROTOTYPE>
    <Short-Name>LaneDetected</Short-Name>
    <Type-TRef Dest="IMPL-DATA-TYPE">/genivi/aasr/showcase/LaneType</Type-TRef>
  </VARIABLE-DATA-PROTOTYPE>
  <EVENTS>
</SERVICE-INTERFACE>

* ARXML is shortened for the presentation
Interoperability Issue during the Prototype

<table>
<thead>
<tr>
<th>vsomeip as the Common Basis</th>
<th>Example: String</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common API</strong></td>
<td><strong>byte order mark</strong></td>
</tr>
<tr>
<td>capi someip RT</td>
<td>String</td>
</tr>
<tr>
<td>vsomeip binding</td>
<td>‘\0’</td>
</tr>
</tbody>
</table>

Byte Order Mark and Terminating ‘\0’ were not considered in the AUTOSAR Platform Demonstrator

https://jira.autosar.org/browse/AR-68397
Feel free to visit the demonstrator during the GENIVI Showcase & Reception!

CES, Las Vegas, January 2019

European R-CAR Consortium Forum, Düsseldorf, March 2019
CONCEPT MAPPING ISSUES and MITIGATION
Marc BELLANGER, Renault Software Labs
CONCEPT MAPPING ISSUES

Some specific concepts do not find an equivalent on the other side.

Selective Broadcast
Polymorphic structures
Interface version
Optional fields
Data semantic
Method errors

- Allow to send broadcast to dedicated client but against SOA paradigm where only middleware knows the registered client list.
- CAPI uses a TV serialization with hash value for the TAG. But not defined in SOMEIP specification
- No version in ARXML for interface definition. (managed at SOMEIP deployment level)
- Introduced in AUTOSAR ADAPTIVE 18.10. Field is present if Tag is present in TLV serialization format. CAPI serializes structs with LV (length presence and width is configurable in FDEPL)
- Used to define the content of an unitary data. (Unit, max value...)
- Not defined in FIDL.
- Application errors are not transported on the same way. Autosar uses SOMEIP Error code to transport applicative errors. CAPI generated code define a mandatory error status.
Tracing with DLT and EB solys

Torsten Mosis, systemticks GmbH
Tracing with DLT and EB solys

- Trace SOME/IP message calls into DLT
  - Non-intrusive via network packet sniffing
  - With instrumented code (e.g. configured and generated through Franca Deployment Models)
- Analysis with EB solys
  - Connect to dlt daemon
  - Decode SOME/IP messages into human readable text
  - Trace back and map to origin Franca Models
  - Check method call integrity
  - Validate right orders of messages
  - Show dependency graph
NEXT STEPS

Marc BELLANGER, Renault Software Labs
Next steps

**CONFIRMED**
- Tool stage 2
  - Near production level

**NICE TO HAVE**
- Align the deployment information.
  - FDEPL (COMMONAPI) vs. ARXML (AUTOSAR)
  - Ease the Deployment of SOMEIP with CAPI (Generate VSOMEIP.json from FDEPL ?)

**OPEN**
- Compatibility with ANDROID JAVA applications ?
Thank you!

Visit GENIVI:
http://www.genivi.org
http://projects.genivi.org

Contact us:
help@genivi.org