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

GENIVI®

Agenda

11:00 - 11:15	**Motivation**	Introduction
		Objectives
11:15 - 11:40	**Model level**	Mapping of models
		Transformation tooling
		Q&A / discussion
11:40 - 11:55	**Runtime**	Demonstrator - from CES to AMM
11:55 - 12:15		Concept mapping issues and mitigation
		Q&A / discussion
12:15 - 12:20		Logging & Tracing
12:20 - 12:30	**Next steps**	

GENIVI®

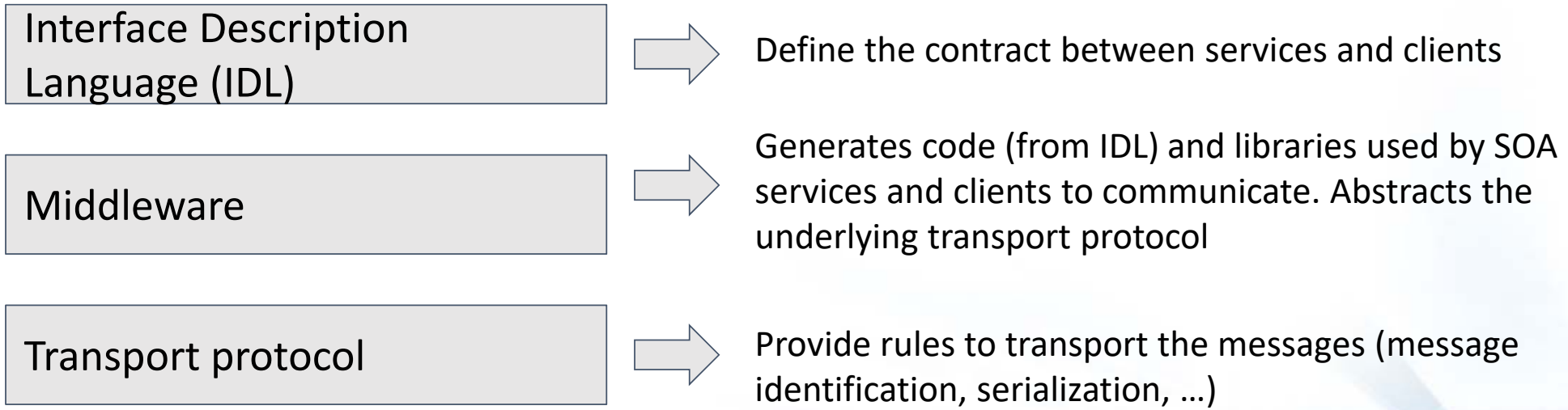
INTRODUCTION

Marc BELLANGER, Renault Software Labs

GENIVI®

Introduction

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



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 ?**

	GENIVI	AUTOSAR
IDL	FRANCA IDL	ARXML
Middleware	COMMONAPI	ARA::COM
Transport protocol	SOME/IP	SOME/IP

OBJECTIVES

Marc BELLANGER, Software Architect, Renault Software Labs

GENIVI®

Objectives

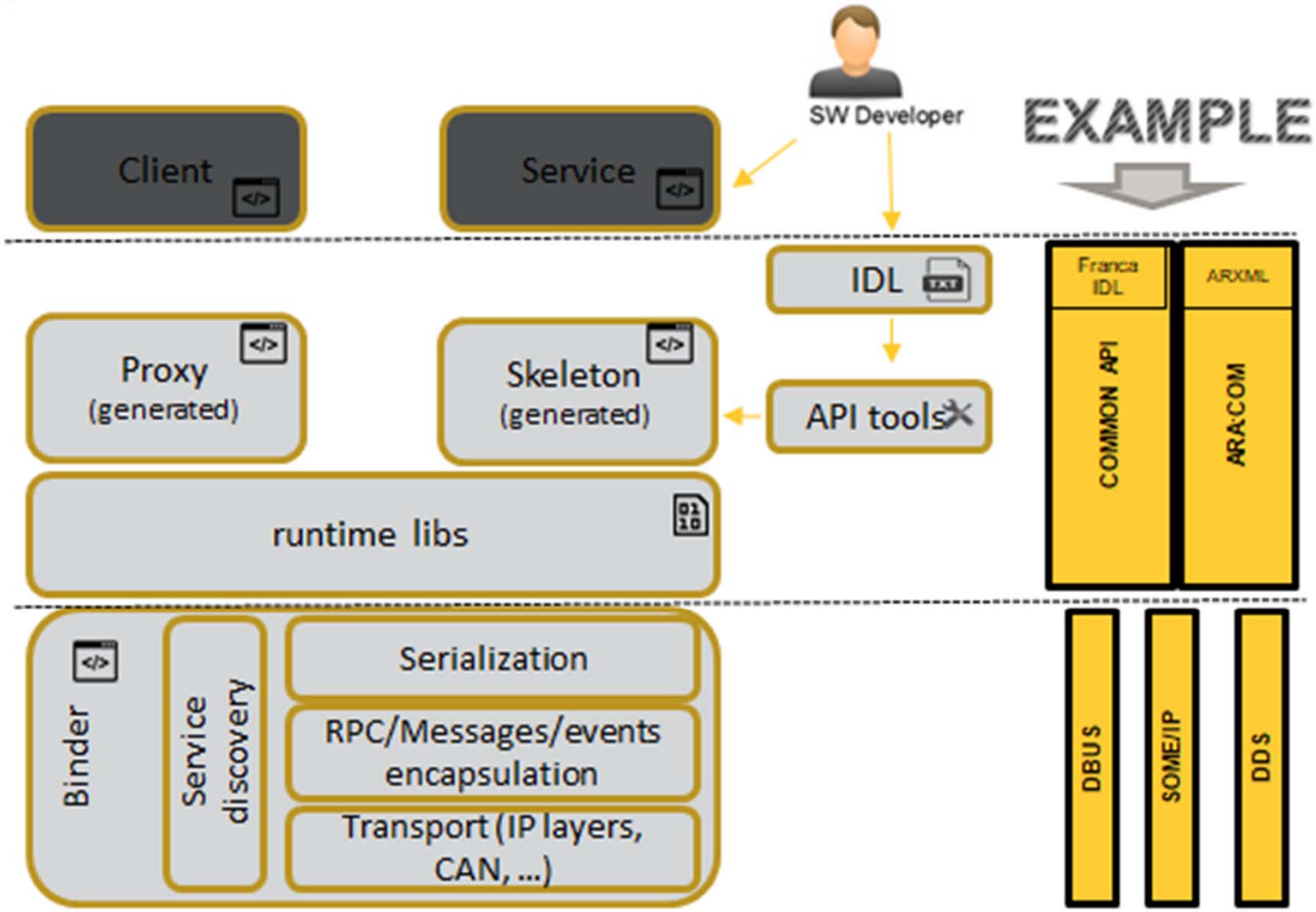
To ensure compatibility we have to reach 2 objectives

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

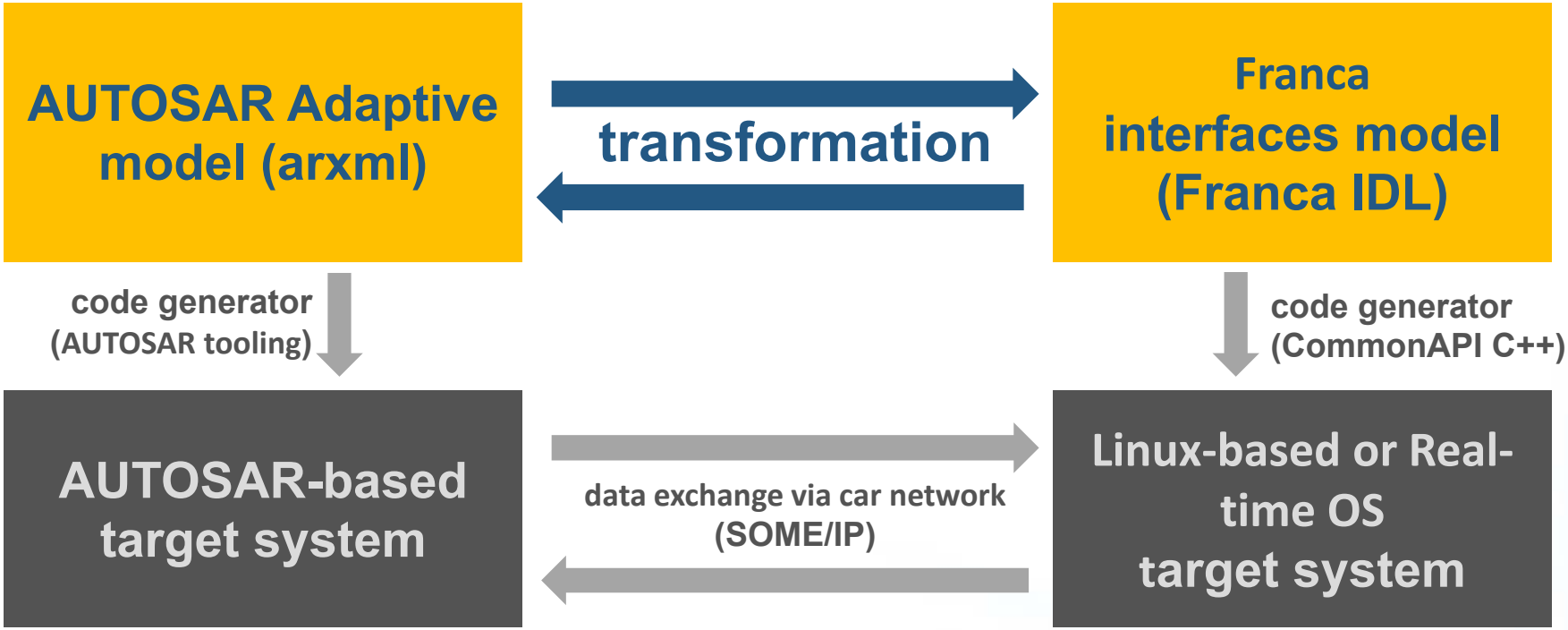


Model-level Mapping of AUTOSAR Adaptive and Franca IDL

Dr. Klaus Birken, itemis AG

GENIVI®

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)

Mapping table

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

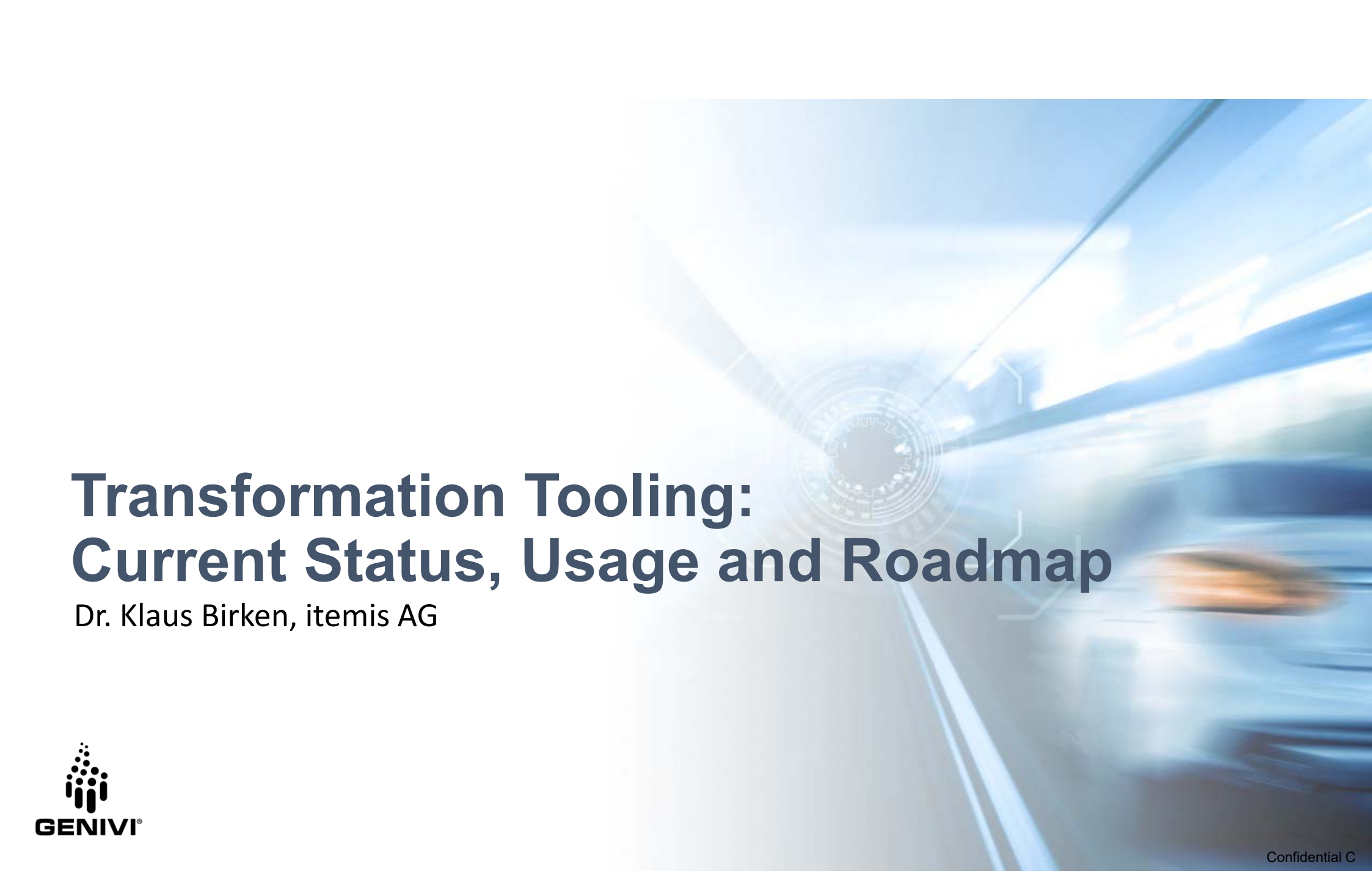
Completed: 80,1%

Group	Franca concept or one of (IGNORE I ERROR I EMULATE) (see A10)	Franca metamodel export ID	Franca metamodel classifier	Franca metamodel attribute	AUTOSAR concept or one of (IGNORE I ERROR I EMULATE) (see A10)	Artop metamodel classifier	Artop metamodel attribute	Detail level (IDL, Serialization, CommonAPI, SOMEIP)
structure	version of typpe 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<PackageableElement> 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<Field> getFields()	
structure	list of methods	IDL1240	Class FInterface	List<FMethod> methods	client server operations of a service	ServiceInterface	List<ClientServerOperations> 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> managedInterfaces	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()	

- read-only link to document on GoogleDocs: [mapping table]

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

GENIVI®

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

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 ARAConnector
val fromFranca = conn.fromFranca(fmodel) as ARAModelContainer
conn.saveModel(fromFranca, "result.arxml")
```

Model transformation tooling: Roadmap

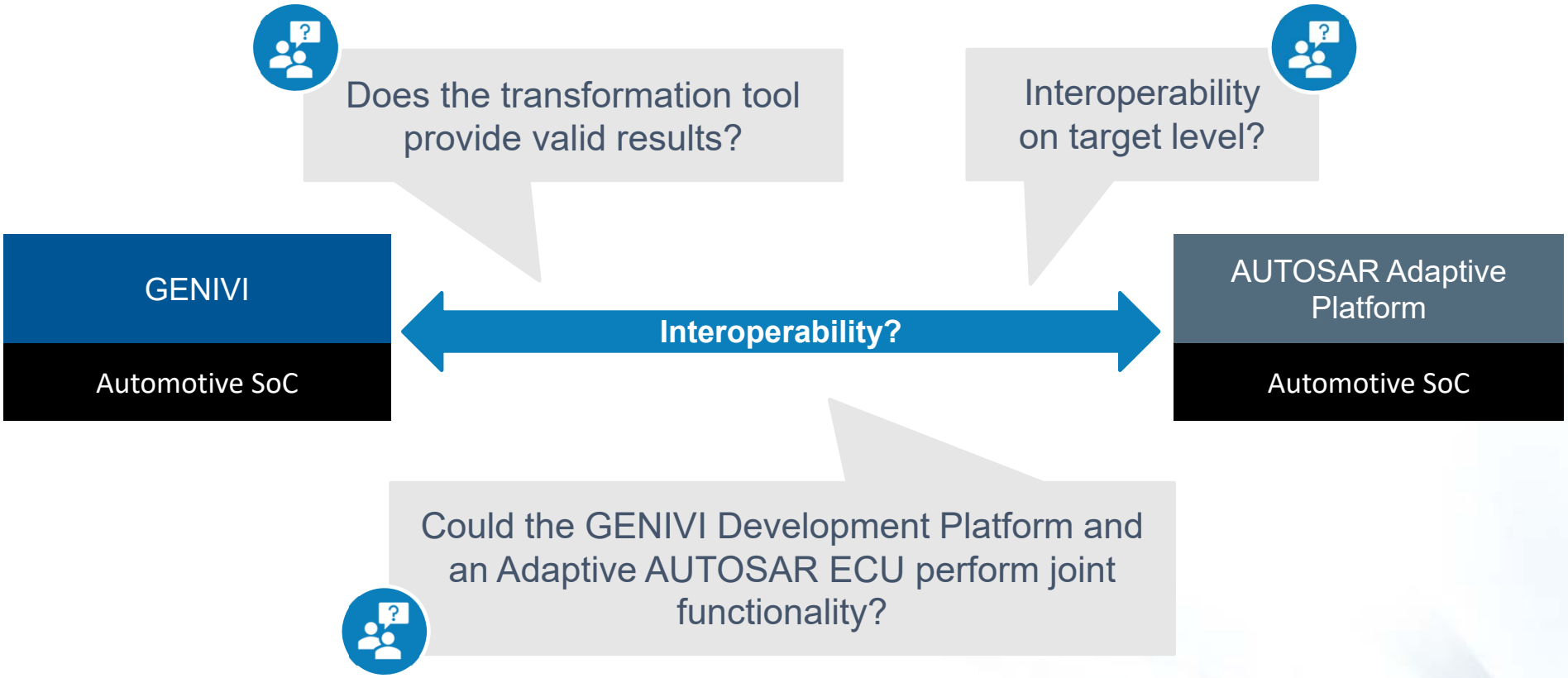
- project start stage 2: started May 2019
- tasks
  - actual transformations (both directions)
  - test cases based on simple and real-world models
  - analysis of SOME/IP deployment mapping
  - automatic build of the tools
  - installable features, command-line tool
- releases for beta testers: continuously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

Demonstrator – from CES to AMM

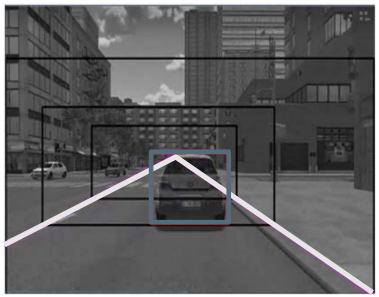
Christopher Schwager, Senior Expert Embedded Architectures, ITK Engineering GmbH

GENIVI®

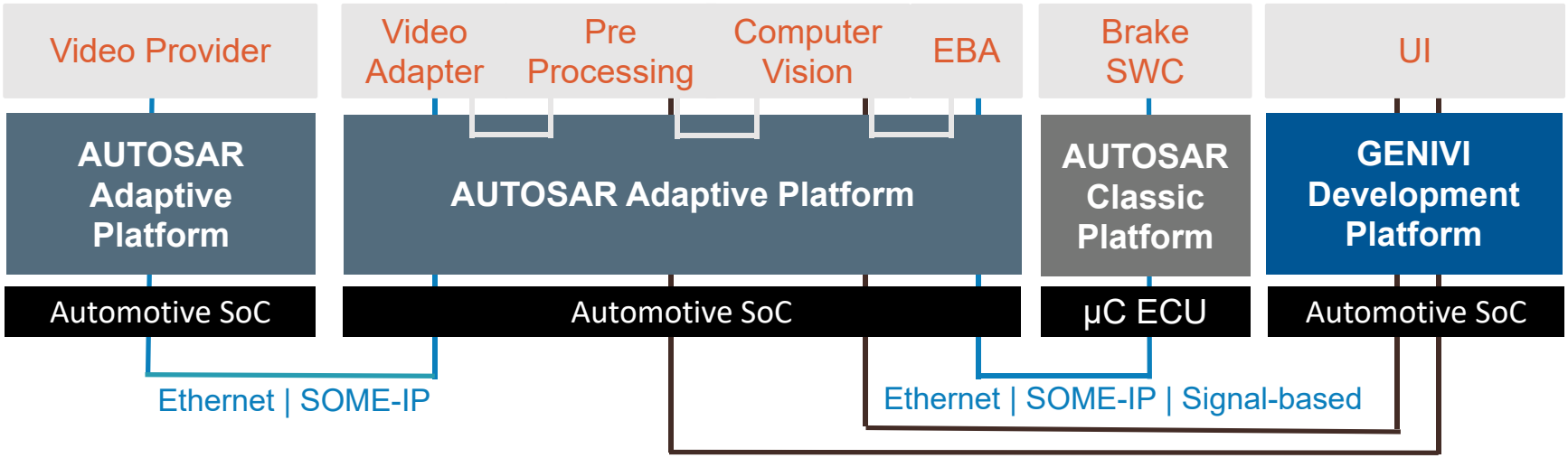
Demonstrator – from CES to AMM



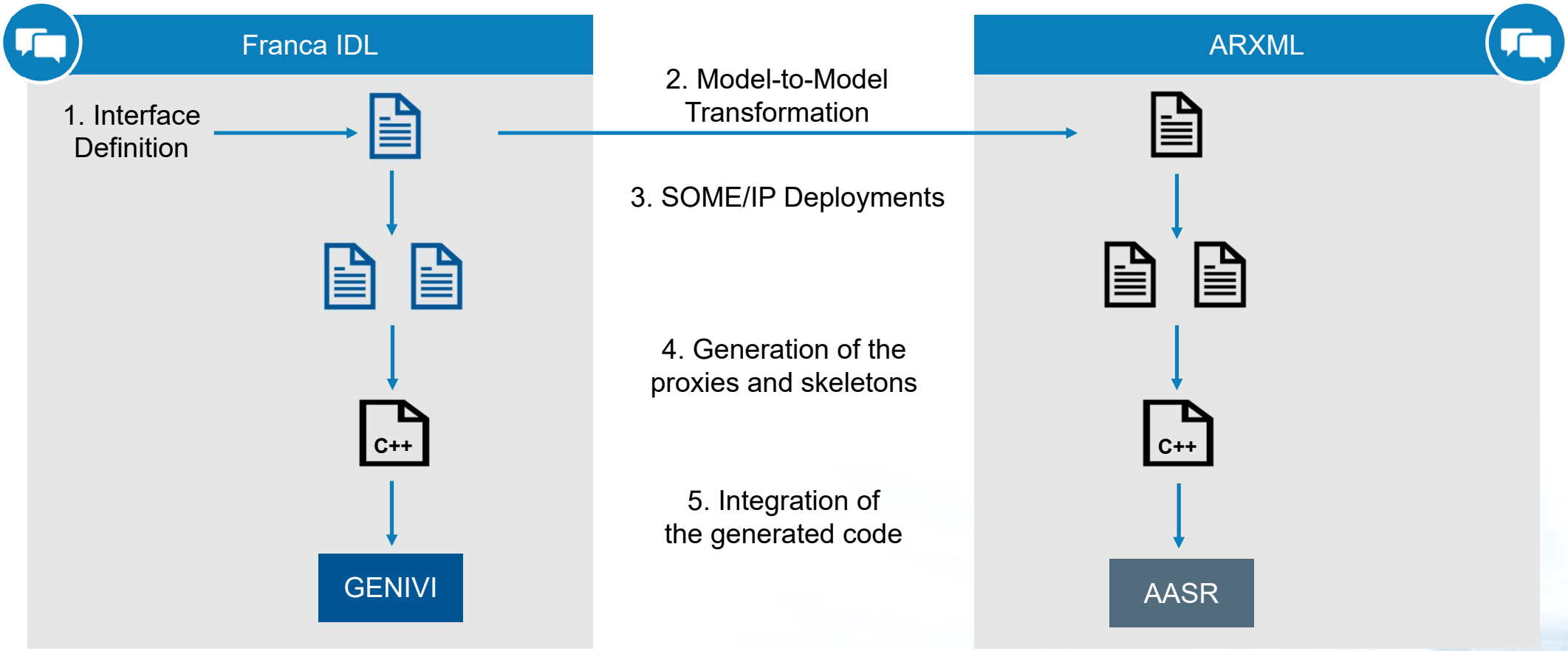
Emergency Brake Assistant Scenario



Emergency Brake Assistant



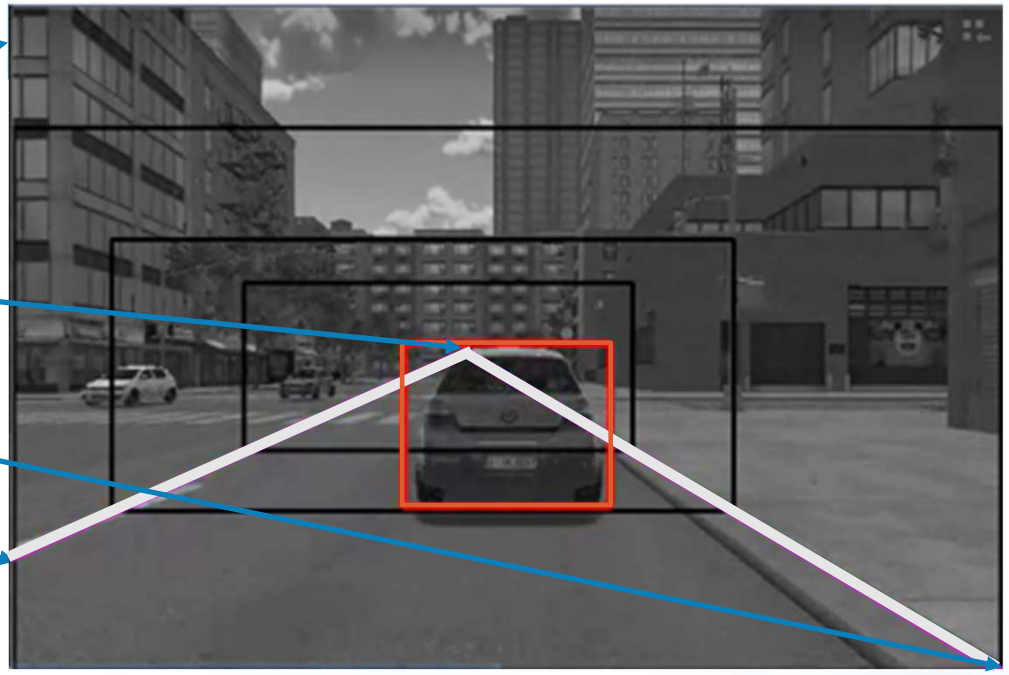
Steps Executed to Establish the Communication



Deep Dive – Driving Lane Interface

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
     }
   }
}
```



Deep Dive – Driving Lane Interface Cont‘d

```
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
    }
  }
}
```

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>LaneType</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>frameId</SHORT-NAME>
      <CATEGORY>TYPE_REFERENCE</CATEGORY>
      <IMPL-DATA-TYPE-REF …>/ara/stdtypes/UInt16</IMPL-DATA-TYPE-REF>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>lowerLeftPointX</SHORT-NAME>
      <CATEGORY>TYPE_REFERENCE</CATEGORY>
      <IMPL-DATA-TYPE-REF …>/ara/stdtypes/UInt32</IMP-DATA-TYPE-REF>
      </IMPLEMENTATION-DATA-TYPE-ELEMENT>
    ...
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

* ARXML is shortened for the presentation

Deep Dive – Driving Lane Interface Cont‘d

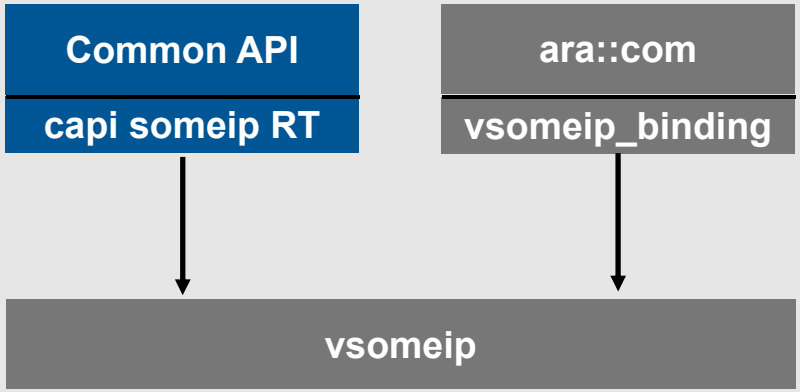
```
package genivi.aasr.showcase

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

vsomeip as the Common Basis



Example: String

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

GENIVI®

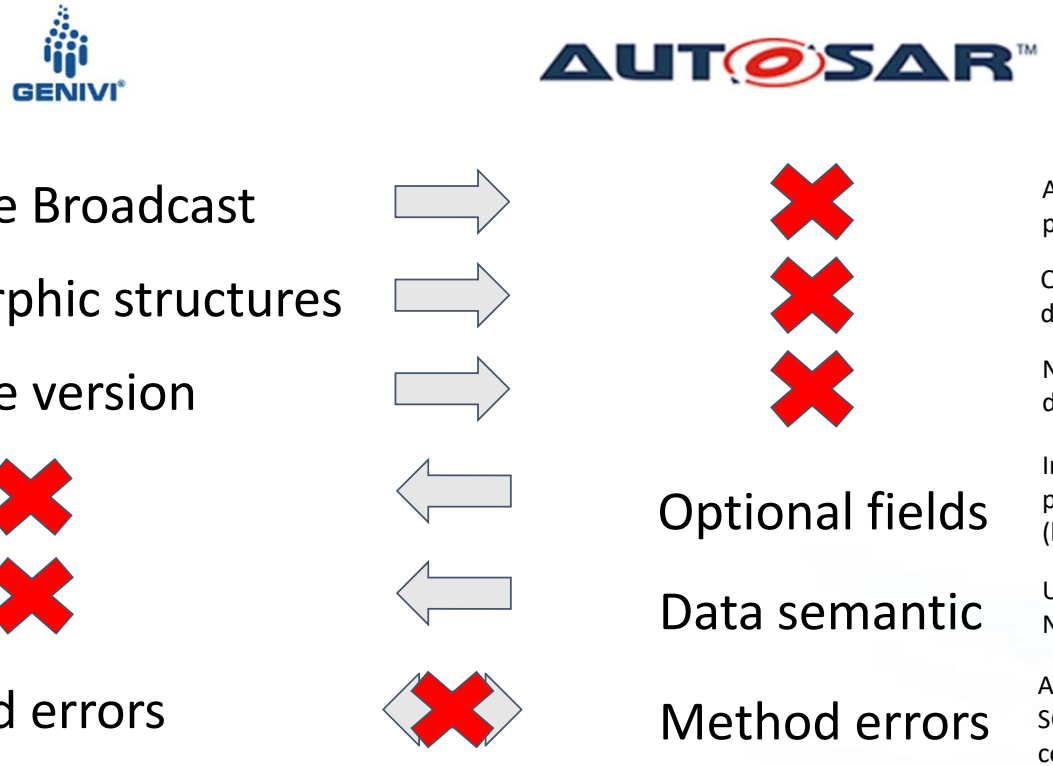
CONCEPT MAPPING ISSUES and MITIGATION

Marc BELLANGER, Renault Software Labs

GENIVI®

CONCEPT MAPPING ISSUES

Some specific concepts do no find an equivalent on the other side.



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.

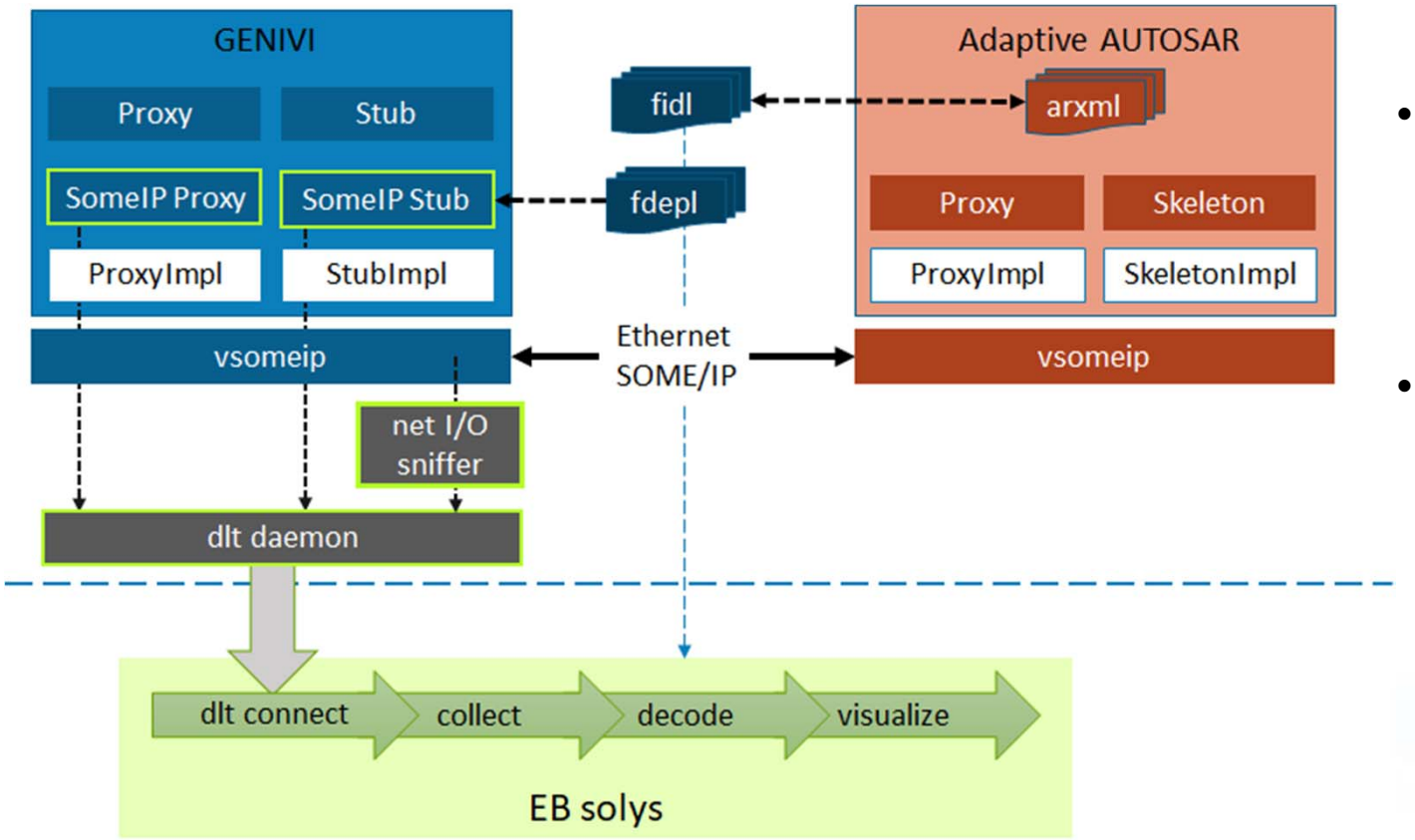
Application errors are not transported on the same way. Autosar uses SOMEIP Error code to transport applicative errors. CAPI generated code define a mandatory error status.

Tracing with DLT and EB solys

Torsten Mosis, systemticks GmbH

GENIVI®

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

GENIVI®

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

GENIVI®