A Semantic Analysis of Interface Description Models of Heterogeneous Vehicle Application Frameworks: An Approach Towards Synergy Exploration

Sangita De[1,3], Michael Niklas[1], Jürgen Mottok[2] and Přemek Brada[3]
[1]*Corporate System & Technology SW, Continental Automotive GmbH, Regensburg, Germany*
[2]*Department of Electrical Engineering & Information Technology, OTH, Regensburg, Germany*
[3]*Department of Computer Science and Engineering, Universitsy of West Bohemia, Pilsen, Czech Republic*

Keywords: Application, Framework, IDL, Semantic, Mapping, Traits, Synergy, Metamodel, Syntax, Component Model, Domain, Vehicle, Interface, Messages, Fields, RPC, Client-Server, Analysis, Structures, Port, Service, IPC.

Abstract: As the world is getting more connected, the demands of services in automotive industry are increasing with the requirements such as IoT (Internet of Things) in cars, automated driving, etc. Eventually, the automotive industry has evolved to a complex network of services, where each organization depends on the other organizations, to satisfy its service requirements in different phases of the vehicle life cycle. Because of these heterogeneous and complex development environments, most of the vehicle component interface models need to be specified in various manifestations to satisfy the semantic and syntactic requirements, specific to different application development environments or frameworks. This paper describes an approach to semantic analysis of components interfaces description models of heterogeneous frameworks, that are used for vehicle applications. The proposed approach intends to ensure that interface description models of different service-based vehicle frameworks can be compared, correlated and re-used based on semantic synergies, across different vehicle platforms, development environments and organizations. The approach to semantic synergy exploration could further provide the knowledge base for the increase in interoperability, overall efficiency and development of an automotive domain specific general software solutions, by facilitating coexistence of components of heterogeneous frameworks in the same high-performance ECU for future vehicle software.

1 INTRODUCTION

A semantic analysis on domain specific applications (apps) of heterogeneous frameworks (FWs) can be dedicated to specific kind of artefacts (e.g. deployment possibilities of app component models, etc) or it can be general and capable of including any constituent of the metamodel ecosystem in context of app software. The Interface Description Language (IDL) model is an integral part of an app FW and is represented using a platform specific language. The IDL model of a Service Oriented Architecture (SOA) based vehicle app FW, usually describes the services that are offered or required by an app in an abstract form, independent of implementation details. Every vehicle app IDL model can be expressed in two basic forms. Firstly, FW specific language notation, named as syntax, and secondly, meaning of the syntax named as semantics. Syntax of an IDL is possibly infinite set of legacy elements, and is augmented by the meaning of those elements, which is expressed by relating the syntax to a semantic domain (Weinreich and Sametinger, 2001). Therefore, any app FW IDL definition must consist of the syntax domain and mapping from the syntactic elements to the semantic domain. The approach to synergy exploration using semantic comparison of interface models of vehicle components can be used to find the correlation among the interface syntax of these components.

1.1 Contribution of the Report

A vehicle interface description model usually has commonality in semantics, despite using different IDLs, when modelling the same app for a vehicle component in different FWs. However, the SOA based vehicle FWs have IDLs which differ mostly in concrete syntax representations because their manifestations are adapted to a specific app FW to

De S., Niklas M., Mottok J. and Brada P.
A Semantic Analysis of Interface Description Models of Heterogeneous Vehicle Application Frameworks: An Approach Towards Synergy Exploration.
DOI: 10.5220/0007472503960403
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 7th International Conference on Model-Driven Engineering and Software Development)*, pages 394-401
ISBN: 978-989-758-358-2

which they are integrated. This further results in inefficient vehicle app software component reuse and increase in overall development cost. The gaps between these IDLs of heterogeneous vehicle app FWs, can be reduced by analysing the semantic synergies in semantic mappings of their interface models, thereby leading to more efficient vehicle app software component reuse and reduction in overall development cost.

The goal of this paper is to compare the semantics of various SOA based vehicle app FW interface models using mappings and to explore the synergies in semantic mappings. This will help future domain experts to understand the semantic synergies of interface models and decide which semantic synergies could be considered for creating any kind of domain specific general software solutions such as a Meta Interface description model for automotive domain. In the future, semantic mappings of interface models of vehicle FWs could be further used for translation of semantics required for app component model transformation from one FW to another FW (Ruscio et al., 2012).

1.2 Motivation Scenario and Related Work

An app component model description is always useful for exchange of information between the FW experts of the given component model. The specifics of an app component model however make it quite difficult to read and understand the architecture of app component for experts from other different vehicle FWs (Brada and Snajberk, 2011). Over the past few years, the demand for cross sub-domain functionalities in the automotive domain has increased. Consequently, it has become necessary to combine the software components and subsystems as well as message formats from different sub-domains, to provide cross domain functionalities (Avram et al., 2014). This could be due to the fact: firstly, the increase in requirement for integration with 3rd party and legacy components, secondly, the conformance to frequent new standards in automotive domain (Birken, 2013), thirdly, the non-functional system requirements such as performance and footprints and lastly, the requirement of huge number of communicating processors for cross sub- domain communications. Therefore, it is required to cluster the vehicle apps based on a software functional area and glue the relevant artefacts between them (Pretschner et al., 2007). This can be done by identifying the synergies in semantic mappings among the vehicle app IDL models of an app cluster.

There are several IDLs of SOA based FWs which are used for vehicle app interface model specifications. The OMG (Object Management Group) standard has an open distributed object computing infrastructure CORBA (Common Object Request Broker Architecture) (Gokhale et al., 2007). CORBA provides object services that are service interfaces to be used by many distributed object programs regardless of application domains, but CORBA faces performance trade-offs with high speed networks. Franca FW provides special support to those automotive domain IDLs which can be implemented using EMF (Birken, 2013). Based on the Franca core model, service interface specifications defined in other IDLs can be transformed to or from Franca. However, there are still some unanswered questions like successful extension of Franca connectors to messages. These questions could be addressed using semantic mapping analysis.

Message component interface models from ROS (Robot Operating System) and Google Protobuf (Protocol buffers) used for different message serialization and deserialization purposes during message transmission and reception in vehicle apps, have several pros and cons. There are proposals for establishing a bridge between ROS and Protobuf FWs to overcome their cons and to obtain a merged support from both ROS and Protobuf IDLs (Dhama, 2017). To establish this kind of bridge basically requires statical analysis of metamodels to understand the semantic mapping and synergies between given specific FWs app interface models.

1.3 Automotive IDL: The Rationale

In domain specific models such as automotive system models, the metamodels are often originally introduced as structuring elements. These elements give semantics to traditional modelling languages and thereby also describes semantics for interface models (Ruscio et al., 2012). The Figure 1 illustrates overview of heterogeneous software components (SWCs) supported by different app FWs say A and B and supported by different Operating Systems (OSs). In context of a concrete app, a software component implementation must conform to one of the component types defined by its component model. An app software component model has a set of concrete interface elements manifest on the visible surface of its black box. These model elements populate some or all its actual traits, which again conform to the corresponding trait definitions.

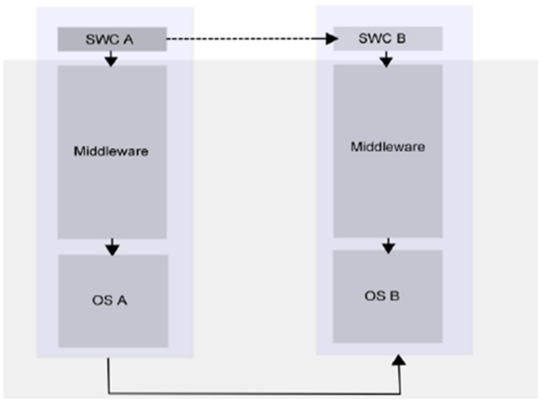


Figure 1: Overview of interfaces of SWC.

An interface not only specifies the services, call-backs or function calls that a client may request from a service provider app component, but it may also include constraints on the usage of these services that must be considered by both the service app component and its clients. Interfaces are service contracts between a service provider and a service receiver. This contract may include numerous invariants, preconditions and post conditions such as deployment conditions of an app component interface model that must be considered when using an individual operation.

Most automotive domain FW component models have an IDL for describing interfaces and their elements using an implementation independent semantic and syntactic notation (Lau and Wang, 2007). In context of SOA based FWs that are mostly used in automotive domain, an app IDL model basically includes information on following elements at an abstract level:

- Service definition using signals or messages, structure or events and broadcasts specifications;
- A unique name for the service or identity for the interface;
- Method signatures containing Semantic and Syntactic Information with valid parameter types, e.g. methods used for Service subscription (or registration), Service publication, Service notifications (using callback notifications), etc.;
- Attributes (or member variables of an interface), fields and Data Types (e.g. primitive, complex);
- Optional deployment features based on supported middleware or communication protocol used by different app FWs.

2 SEMANTIC ANALYSIS OF IDL MODELS: THE APPROACH

The problem to find the differences in the component's interface model of heterogeneous FWs is intrinsically complex and requires specialized algorithms to match the abstraction levels of models.

2.1 Static Semantic Analysis

Based on explicit semantic content authoring or static analysing, approaches can be roughly classified into two basic categories Top-Down and Bottom-Up (Brada and Snajberk, 2011). The manual static semantic analysis approach considered in the current scope is a Top-Down approach and is at an early stage, without the use of any automated static analysis tool. The approach is based on the starting point of an authoring process which is upper most level of expressiveness. The proposed approach considers Interface_basic_type of the interface specification as the starting point or trait for analysis process. The current approach defines the abstract traits of a vehicle FW's component interface model for the semantic analysis. The approach further uses semantic mapping to compare these traits for few of the existing IDL alternatives used by vehicle FWs. The approach uses a common case study of *SeatHeating* SWC to realize an abstract interface model by using each of the IDL alternatives. The *SeatHeating* SWC is a sensor actuator component model used in vehicle to monitor seat heat.

2.2 Classification of Traits for Semantic Analysis

The abstract functional traits that are required for a vehicle app interface model, have been classified based on SOA based FW's component metamodels basic features. For the traits, the metamodel representation ∈ Identifiers defines a trait's metamodel representation name, and metatype ∈ Identifiers defines type of the trait in context of metamodel (Brada and Snajberk, 2011). The specifications of abstract functional traits for component interface model elements are provided below, where comm stands for communication:

- Interface_basic_type
  metamodel representation: basic_intf_element
  metatype:basic_element_specification;
- Service_Interface_connection_point
  metamodel representation: service_connect_pt
  metatype: Interface_ports;

- Intf_bind_comm_proto
  metamodel representation: comm_proto
  metatype: Interface_binder_comm_protocol;
- Communication_Method_Specification
  metamodel representation: method_spec
  metatype: Interface_communication_method;
- Method_behaviour_Specification
  metamodel representation: method_behav
  metatype: asynchronous,synchronous;
- Field_Specification
  metamodel representation: field_spec
  metatype: attributes;
- Records_Specification
  metamodel representation: record_spec
  metatype: Datatypes;

Traits for an IDL can also represent non-functional elements. The specifications of non-functional traits for component interface model are provided below:

- Versioning: Interface compatibility;
- Software Licence supported;
- Language Bindings supported.

3 IDL MODELS OF VEHICLE APP FWS: ALTERNATIVES

This section provides an overview of abstract interface models using few of the existing IDL alternatives of heterogeneous FWs used for vehicle apps (Dhama, 2017). The abstract interface models discussed in this section are based on *SeatHeating* SWC model case study, as described in subsection 2.1.

3.1 Franca IDL

Franca IDL is developed as a part of the GENIVI standard Franca FW and supports IVI (In-Vehicle infotainment) system's interfaces. Franca IDL is language-neutral and independent of concrete bindings. APIs (Application Program Interfaces) defined with Franca IDL consist of collections of attributes, methods and broadcasts (Birken, 2013). Primitive datatypes supported are (Un)signed integers, Float/Doubles, Strings and Byte Buffers. Using Franca IDL, a vehicle app client calls the backend server using a vehicle ID and a struct (Structure) defining service information such as a unique Service Id. Figure 2 illustrates *SeatHeating* vehicle app using Franca IDL abstract model. With Franca IDL, a vehicle app is free to use older versions of service interface methods with newer versions of Franca IDL models. Therefore, in context of versioning, Franca IDL has backward compatibility.

```
/* Structure grouped under
Services*/
interface SeatHeatingServ {
version (major 2 minor 0),
struct HeatingElementService
{UInt32 ServiceId, String
ServiceName}
/* Method specification*/method
subscribeSeatHeatingServ{
in {UInt32 VehicleId
HeatingElementService myService}}}
```

Figure 2: Abstract model of Franca FW Interface with Service Structures and Methods.

Franca+ provides an extension to the Franca FW that adds support to the modelling for software and hardware components. It uses the same textual language style as Franca but supports the definition of components, composition of components, typed ports and connectors between ports as seen in the Figure 3.

```
/*A Client Component for Heating
Element */
Service component SeatHeatingControl
{   requires SeatHeatingElement as
AnswerMePort
    } /* The Server Component for
Heating Element */
Service component SeatHeating {
    provides SeatHeatingElement as
AskMePort}
```

Figure 3: Abstract model for Franc++ component Interface.

The "provides" keyword, within a component definition, is used to define a provider port. Similarly, the "requires" keyword is used to define a required port. Franca+ plans to extend the Franca interface deployment model based on RPC (Remote Procedure Call) mechanism. The Common API provides the middleware solution supported by GENIVI as a part of Franca FW. With Franca Interface model, an app communicates with the Common API library and not with the IPC (Inter Process Communication) directly thereby making the app IPC agnostic.

3.2 Google Protocol Buffers IDL

Protobuf are a flexible, efficient, automated mechanism for serializing structured data, used in IVI systems e.g. vehicle telematics data exchange, etc. Protobufs are open source since 2008, prior to that they were internally used by Google (since 2001). A strong aspect of Protobuf data descriptions is the ability to update, in a backward compatible and forward compatible way, without affecting the already deployed systems. Versioning is done by using unique field numbers. The meta-data in Protobuf is in form of key-pairs. In contrast to Franca IDL, the base of defining interfaces within Protobuf is through the definition of messages.

Messages are identified by a name and contain fields, each with a unique field number as illustrated in an example in Figure 4 (Dhama, 2017). However, like Franca IDL Protobuf also uses a RPC mechanism known as gRPC (Google RPC) for deployment of message component models. RPC binding done via gRPC generates stubs and skeletons. The Search Request for service or Search Response messages (basically a structure) contains 2 essential fields as per proto3 syntax, illustrated in Figure 4. Fields can either be:

- *singular*: with multiplicity 0...1. No keyword is needed in this case;
- *repeated*: with multiplicity 0...N. Keyword *repeated* is used. It is used to define an array type.

```
/*Search request for Interface
SeatHeating Operation */
message SeatHeatingOperationRequest
{
string parameter1 = 1; /* Singular
Field Specification */
bool parameter2 = 2;}
/*Search response for Interface
SeatHeating Operation */
message
SeatHeatingOperationResponse {
bool parameter2 = 1; repeated
string parameter3 = 1;}
/*Repeated Field Specification*/
Service SeatHeatingserv{rpc
operation(SeatHeatingOperationReque
st);returns(SeatHeatingOperationRes
ponse);}
```

Figure 4: Abstract model of service interface messages using Google Protobuf.

gRPC is a high performance open source RPC FW. Client machines can seamlessly call server machines as if they are in the same execution environment. Protobufs are designed for messages that are 1MB in size or smaller. Therefore, Protobuf by default, will not deserialize a message larger than 64MB. Protobuf uses two different kinds of message transport mechanisms. When transferring of client(stubs) to server (skeleton) RPC messages within a single machine, Protobuf uses IPC using shared memory and event based synchronizations. When there is a requirement to transfer RPC messages from client or subscriber to server or publisher including multiple host machines, Protobuf uses inter host communication based on UDP (Unified Datagram Protocol) multicast feature.

3.3 Apache Thrift IDL

Thrift is a software library and set of code-generation tools developed at Facebook to expedite development and implementation of efficient and scalable backend services. Thrift, which is supported by Apache Software Foundation standard, allows developers to define datatypes and service interfaces in a single language-neutral file. Thrift generates all the necessary code to build RPC clients and servers that communicate seamlessly across various programming languages (Slee, Agarwal and Kwiatkowski ,2007). and is frequently used in the vehicle apps e.g. monitoring of driver behaviour apps using sensors.

Apache Thrift IDL is a superset of Protobufs, with additional features that does not exist in Protobuf such as constants, rich containers types e.g. list, maps, sets, etc. The RPC invocation is done by sending a method name on wire as string. Thrift defines interfaces using Structures. Struct (structures) are grouped under services like Franca IDL. An example of *SeatHeating* service to track the seat heat using Thrift IDL is shown in Figure 5. The field header for every member of a struct is encoded with a unique field identifier.

```
/*Interface using structures*/
struct SeatHeatingElement {
1: required double seat_temp;
2: required double heating_calib;
};
/*exception*/
exception DBUnavailable {
1: string ErrorCode;};
/*Service Specification*/
service SeatHeatingserv {
bool updatetemp(1:
SeatHeatingElement elem) throws
(1: DBUnavailable naService);}
```

Figure 5: Abstract model for thrift struct and services.

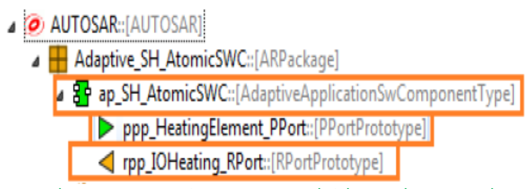
The Thrift IDL also supports versioning. Versioning in Thrift is implemented via field identifiers Thrift is robust to versioning and data definition changes. Thrift supported app FW must be able to support requests from out-of-date clients to new servers, and vice versa. Therefore, in context of versioning, Apache Thrift IDL has forward & backward compatibility like Franca IDL.

3.4 AUTOSAR XML (ARXML)

ARXML (AUTOSAR eXtensible Markup Language) is the standard description format used to model all AUTOSAR (Automotive Open System Architecture) software component models related to the AUTOSAR Classic platform and the AUTOSAR Adaptive platform. AUTOSAR is widely accepted as the de-facto standard of automotive system software architecture for developing automotive app of various automotive platforms during the different phases of a vehicle life cycle. The AUTOSAR app software component (SWC) template meta model is implemented using ARXML Schema (Dhama, 2017).

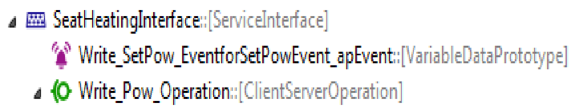
One of the major benefits of using ARXML as IDL is to simply the comparison of AUTOSAR SWC descriptions from different AUTOSAR based automotive platforms. This further enables interoperability among different AUTOSAR platforms such as AUTOSAR Adaptive and AUTOSAR Classic. Figure 6 illustrates an *SeatHeating* SWC model using ARXML. The AUTOSAR SWC have provider port (PPortPrototype) and receiver port (RPortPrototype) interfaces like Franca+. The app software component uses service interfaces. An example of AUTOSAR Adaptive app software component release version 4.0.3 specific ARXML file can be seen in Figure 6.

```
/*Software Component Model
Specification*/
```



```
AUTOSAR::[AUTOSAR]
  Adaptive_SH_AtomicSWC::[ARPackage]
    ap_SH_AtomicSWC::[AdaptiveApplicationSwComponentType]
      ppp_HeatingElement_PPort::[PPortPrototype]
      rpp_IOHeating_RPort::[RPortPrototype]
```

```
/*Service Interface specification using
Methods and Events */
```



```
SeatHeatingInterface::[ServiceInterface]
    Write_SetPow_EventforSetPowEvent_apEvent::[VariableDataPrototype]
  Write_Pow_Operation::[ClientServerOperation]
```

Figure 6: SeatHeating vehicle app SWC model implementation using ARXML.

The Service interface model uses RPC communication protocol, like Franca, Protobuf and Thrift IDLs. Service interface is specified using various elements (AUTOSAR AP release, 2010), these includes:

- Aggregation of variable data prototypes in the role of *Events*;
- Aggregation of meta-class Fields in the role of *Fields;*
- Aggregation of Client-Server Operations in the role of *Methods*.

In AUTOSAR Adaptive platform, a Service Instance Manifest file contains Service deployment description. The services provider SWCs are called skeleton and the service receiver SWCs are called Proxy. For model migration of a AUTOSAR Classic SWC model to the AUTOSAR Adaptive app manifest model, ARXML is used as a common modelling language to represent both source and target SWC models and their interfaces.

3.5 ROS IDL

ROS (Robot Operating System) provides the required tools to easily access sensor's data, process that data, and generate an appropriate response for the motors and other actuators of the robot. Due to these characteristics ROS is a perfect FW for self-driving cars and an autonomous vehicle can be considered just as another type of robot (Berger and Dukaczewski, 2014).

ROS offers a message passing interface that provides IPC and is commonly referred to as a middleware solution. The benefit of using a message passing system is that it forces to implement clear interfaces between the nodes in a system, thereby improving encapsulation and promoting code reuse. The datatypes used by ROS messages is a superset of datatypes used by Google Protobuf and Apache Thrift.

The asynchronous nature of publish/subscribe messaging works for Data Distribution Services (DDS) requirements in robotics, but for synchronous request/response interactions, RPC is used between processes required for higher levels of robot operations. In ROS1 FW, a Master stores topics and service registration information for all other ROS nodes. An example to create a ROS1 FW service node for *SeatHeating* SWC using a node handler for invocation of RPC is illustrated in Figure 7.

```
/* Service node created by Server*/
ros::init(argc,argv,"add_two_ints_s
erver");
ros::NodeHandle n;
ros::SeatHeatingServ service =
n.advertiseService("update_seat_tem
perature", update);
```

Figure 7: Abstract model of a ROS 1 service node using ROS IDL.

To create a client node using ROS1 FW, a ros::ServiceClient object is used to call the service using the same node handler later on as illustrated in Figure 8. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. The nodes can only receive messages with a matching topic type.

```
/* Service requested by Client*/
ros::NodeHandle n;
ros::SeatHeatingControl client =
n.serviceClient<service_start::updat
e_Temp>("update_seat_temperature");
```

Figure 8: Abstract model of a ROS 1 client node using ROS IDL.

4 DISCUSSIONS ON RESULTS OF SEMANTIC ANALYSIS

This section includes tables to illustrate the results from semantical comparison of the various IDL models based on mapping of functional and non-functional traits. Table 1 illustrates the semantical comparison of the various IDL models based on mapping of non-functional interface traits (described in sub-section 2.2): *Versioning* i.e. forward or backward compatibility of a component's interface model, *software Licence supported*, and *Language bindings* used to describe a component's interface model (Ruscio et al., 2012). The fields within the tables marked white indicates semantic synergies in functional and non-functional traits among the different FW IDL model alternatives, using manual static semantic analysis approach. The synergies revealed in the semantic traits comparison of the vehicle FW IDL alternatives, could be further utilized for cross vehicle FW communication of services.

Table 1: Static semantic mapping of FW IDL models based on non -functional traits.

IDL Alter-natives	Versioning	Language Bindings supported	Software Licence supported
Franca IDL	Backward compatibi-lity	C++, C, Java	Genivi Alliance
Protobuf	Forward, Backward Compatibi-lity	C++, Python, Java, C#	BSD
Thrift	Forward, Backward compatibi-lity	C++, Java, Python, Ruby, C#, Perl	Apache
arxml	Backward compatibi-lity	C++, C	AUTOS-AR
ROS IDL	No support to versioning	C++, C, Python	BSD

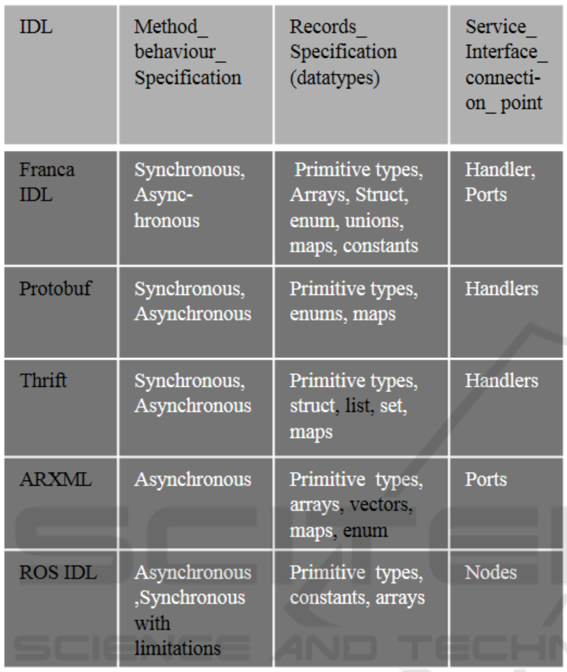
Table 2 illustrates the semantic mapping of vehicle app IDL alternatives based on functional traits such as *Interface basic element type specification or representation*, interface's *Communication Method Specification* used, and the *Communication protocol* that is used for deployment of the FW API models.

Table 2: Static semantic mapping based on functional traits for vehicle component service interface model.

IDL	Interface_ basic_ type	Communica-tion_Method_ Specification	Intf_bind_ comm_pro to
Franca IDL	Structure, Port Interface	Publish-Subscribe, Client.Server	RPC
Protobuf	Messages	Publish-Subscribe	gRPC
Thrift	Structure	Client-Server	RPC
ARXML	Port Interface	Publish-Subscribe	RPC
ROS IDL	Messages	Client-Server, Publish-Subscribe	RPC, DDS IP

Table 3 illustrates semantic synergies based on semantic mapping of the IDLs based on functional traits such as interface *Method behaviour specification, Record specification* used for attributes specification and *Service* (Service Provider or Receiver) *Interface connection point*.

Table 3: Static semantic mapping of FW IDLs based on functional traits.



IDL	Method_ behaviour_ Specification	Records_ Specification (datatypes)	Service_ Interface_ connecti-on_ point
Franca IDL	Synchronous, Async-hronous	Primitive types, Arrays, Struct, enum, unions, maps, constants	Handler, Ports
Protobuf	Synchronous, Asynchronous	Primitive types, enums, maps	Handlers
Thrift	Synchronous, Asynchronous	Primitive types, struct, list, set, maps	Handlers
ARXML	Asynchronous	Primitive types, arrays, vectors, maps, enum	Ports
ROS IDL	Asynchronous ,Synchronous with limitations	Primitive types, constants, arrays	Nodes

Primitive types in Table 3 includes (Un)signed integers, floats, Strings, bytes, Booleans, double.

5 CONCLUSIONS

The paper proposes a manual static semantic analysis approach specifically tailored to explore the synergies in semantics of interface models for vehicle app components of heterogeneous FWs. With the proposed approach, we have defined the abstract functional and non-functional traits as the basic features for a FW component's interface model. We have tried to simplify the semantic comparisons based on the traits, for the various IDL alternatives by using a common case study. Semantic synergies were successfully explored to find the correlation between the IDL models. In the absence of semantic synergy exploration among the IDL models, the translation of the interface semantics of an app SWC model of a given FW to SWC model of another FW is not possible. With the growing demands for services, the functional and non-functional traits considered in the current scope for vehicle FW IDLs, could be further extended for semantic analysis in future. As a proposal for future work, the correlation explored between the different FW IDL models using semantic mappings can be used for any kind of automotive domain specific general software solution such as Meta IDL model. To deal with this, we plan to extend our work of semantic mapping of interface traits in this direction.

REFERENCES

Weinreich, R., Sametinger, J., 2001. *The book*, "*Component Models and Component Services: Concepts and Principles*", G.T. Heineman and W.T. Council (eds.), Reading, MA: Addison-Wesley, pp. 33-48.

Brada, P., Snajberk, J., 2011. "*Ent: A Generic Meta-Model for the Description of Component-based Applications*", Elsevier Electronic Notes in theoretical Computer Science 279(2).

Ruscio, D., Wagelaar, D., Iovino, L., Pierantonio, A.,2012. "*Translational Semantics of a co-evolution Specific language with the EMF Transformation Virtual Machine*", ICMT.

Slee, M., Agarwal, A., Kwiatkowski, M., 2007. "*Thrift: Scalable Cross-language Services Implementation*", *Facebook white paper 5*, 156 university ave.

Dhama, A., 2017. "*Interface Definition Languages", EB white paper,* http://www.elektrobit.com.

Birken, K., 2013. "*Franca User Guide*", Release 0.12.0.1, Eclipse Foundation, itemis AG.

Lau, K., K., Wang, Z., 2007. "*Software Component Models", IEEE Transactions on software Engineering,* Vol 33, Issue 10, pp. 709-724.

AUTOSAR Adaptive Platform (AP) Release, October, 2017. "*Specification of manifest*". http://www.autosar.org.

Berger, C., Dukaczewski, M., 2014. "*Comparison of Architectural Design Decisions for Resource-Constrained Self-Driving Cars-A Multiple Case-Study", in Gesellschaft for* INFORMATIK.

Pretschner, A., Broy, M., Krüger, I., H., Stauner, T., 2007. "*Software engineering for automotive systems*: *A roadmap", In Proceedings FOSS 2007, IEEE Computer Society*, Washington DC, USA, pp. 55-71.

Avram, A., Lenz, D., Bauch, D., Minnerup, P.,2014. "*Interface Definition and Code Generation in heterogeneous Development Enviornments from a Single-Source*", Fortiss GmbH, TMU, Munich.

Gokhale, A., Schmidt, D., C., Ryan, C., Arulanthu, A., 1999. „*The Design and performance of A OMG CORBA IDL Compiler Optimized for Embedded Systems ",* LCTES workshop at PLDI, Atlanta, Georgia.