FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Design Approach to Unified Service API Modeling for Semantic Interoperability of Cross-enterprise Vehicle Applications

The State of the Art and the Concept of Ph.D. Thesis

Sangita De

Technical Report No. : DCSE/TR-2021-02

April 2021

Distribution:Public

Abstract

Heterogeneity among the application component interface data models is an inherent characteristic of open and distributed environments like the automotive domain that hampers interoperability and thus automotive services usage. The automotive industry can be regarded as a complex yet connected network or ecosystem of highly interdependent subsystems. In the recent years with the growth in new era of service requirements for complex automotive services e.g., autonomous driving, Vehicle to Communication (V2X), IoT in cars, etc., there is a subsequent increase in the number of collaborating component frameworks in automotive application domain. With an increase in these vast variety of novel services provided to the passengers; a lot of functionalities are now actively running in parallel on the vehicle's on-board systems supported by complex ECU (Electronic Control Units) software platforms. Data is what all these functions are based on, be it sensor data, user profiles, traffic broadcasts or car-to-car messages from peer vehicles.

The key to success in making such a complex ecosystem work however, lies in the organized and efficient access to the heterogeneous vehicle service frameworks' API (Application Programming Interface) data. In such service collaboration scenarios, most of the API data must be specified in various manifestations to fulfill the specific syntactic and semantic requirements of the service collaborating application frameworks, this further hamper services interoperability and interaction. The collaboration of services between the different frameworks in vehicle application domain and the promotion of the horizontal integration could generate a better and more complete functionality in the automotive industry and open new possibilities.

In today's era, in the absence of a generic, standardized vehicle services API data description template, a source of discord that emerges is that the service providers must always check, before the service deployment, whether the clients or service consumers on the other side of the communication link are compatible with a given service's API. Moreover, a proper service discovery is crucial to identify appropriate services in the growing plethora of third-party services in the vehicle domain. Similarly, from a client's viewpoint, the changes affecting the service collaboration include the retraction of the original service, its replacement by a newer version, support for IDLs to understand the service API syntax and semantics are also essential for interoperability between service provider and service requestor. However, the service oriented automotive software systems still struggle to find means of efficiently checking such compatibility between service APIs. In fact, in the given scenario on service collaboration, identifying semantic synergies between the heterogeneous frameworks' vehicle services API models to achieve the compatibility between the service provider and consumer APIs, looks promising approach.

Interoperability usually refers to the capability that depends on the understanding of compatibility between interfaces of various applications. The incompatibility between the heterogeneous platforms specific artifacts that are part of semantic specification of various vehicle service models causes impediment in semantic interoperability between the API models of the services. From a modeling perspective, to facilitate a holistic and meaningful data exchange between several heterogeneous vehicle services' API models in vehicle domain, one key element to be considered for efficient cross-enterprise services collaborations is to link the platform-agnostic API data at semantic level using a unified, shared vocabulary of the domain. Model transformations and semantic mapping as a part of Model Driven Engineering (MDE) technologies uses new, advanced solutions to address cross-enterprise software interoperability. However, due to the absence of efficient reasoning using domain concepts and inferred artifacts technologies, MDE faces the challenge to achieve interoperability between the semantic concepts of heterogeneous frameworks' vehicle service API models.

As vehicle services are black boxes to the requesters, which means that their source codes are not publicly available. Therefore, to make these services accessible and compatible with one another, their APIs must be specified using standardized, generic semantic specification templates. This research work proposes a design approach to describe the heterogeneous frameworks' vehicle services API models in form of a standardized, platform-independent, generic ontology template. Such a generic API ontology template is expressed in an abstract syntax tree purely based on semantic traits independent of platform details. This generic API ontology template ensures transparency in APIs' semantic data between communication peer partners. To ensure semantic interoperability between the semantically equivalent but syntactically different concepts of various vehicle services frameworks' API ontological models, a platform-agnostic ontology mediator is defined. The proposed ontology mediator is effectively used to glue the semantic bridge between the various concepts of heterogeneous frameworks' service API ontologies, based on identified synergies in semantic traits.

This research contribution uses typical vehicle domain case studies to illustrate the design and implementation approaches towards semantic alignment and integration of heterogeneous vehicle service components' API models.

This work was partially supported by Ministry of Education, Youth and Sports of the Czech Republic, university specific research, project SGS-2019-018 Processing of heterogeneous data and its specialized applications.

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Acknowledgements

I would like to express my deepest gratitude to my supervisors Assoc. Prof. Dr. Přemysl Brada, and Prof. Dr. Juergen Mottok for their continuous support, patience, motivation and sharing of immense knowledge throughout my on-going research study. I would also like to extend my gratitude to my technical supervisor from Continental Automotive GmbH, Mr. Michael Niklas for providing me an opportunity to join the automotive team, his insightful comments and encouragement which made me widen my research from various perspectives. My sincere thanks also go to other Professors from Pilsen university and colleagues from Continental Automotive for their timely cooperation all along the way of my studies. I thank my friends and family for their precious support in all the time of my on-going research study.

Contents

Part I Introduction 13

Chapter 1 Research Goals and Contributions 15

1.1 Research Questions 15

1.2 Overview of Contributions 18

1.3 Report Structure 20

Chapter 2 Background 22

2.1 Role of Interfaces in Interoperability of Vehicle Applications 22

2.2 Fundamentals of Interfaces at Vehicle Software Component Level 24

2.3 Interface Metamodel -The Conceptual Building Block 28

2.4 Event Chain Timing Behavior of Software Components' Interface Models 33

Chapter 3 Related Works 39

3.1 Linking of Heterogeneous and Distributed Data at Semantic Level 39

3.2 Metamodel-based Modeling of SWC's Interface Models 39

3.3 Semantic Mapping of Concepts of Interface Metamodels of SWC Frameworks for Interoperability 40

3.4 Unified API Description/Specification Language for Vehicle Domain Application SWC Frameworks 41

3.5 MDE Vs Ontology Approach for Domain-Specific Interface Metamodels Semantic Alignments: Alternative or Complementary 42

3.6 Evaluation of Interface Metamodels Semantic Alignment Quality for Vehicle Application SWC Frameworks 43

3.7 MDE and Ontology Modeling Approaches to tackle Semantic Interoperability between Vehicle Component Framework Interfaces 44

3.8 Comparison of Author's Contribution to the State of the Art 46

Part II Analysis Level 49

Chapter 4 Survey of Vehicle Domain Interface Description Languages (IDLs): Identification of Semantic Commonalities 50

4.1 Semantic mapping of Component Framework IDLs: The Rationale 50

4.2 Semantic Comparison of Vehicle Domain Cross-enterprise Platforms Component Frameworks IDLs 51

4.3 Technology and Platform Agnostic Specification for Service API Models for Vehicle Domain Heterogeneous SWC Frameworks 60

Chapter 5 Semantic Comparison of Vehicle Component Frameworks' Interface Metamodels 64

5.1 Application Component Framework Interface Metamodels: Alternatives 66

5.2 Summarized Semantic Mapping of Component Frameworks Interface Metamodels 72
Part III Design & Implementation Level (WIP) 73
Chapter 6 Design Approach to Semantic Alignment of Component Frameworks Interface Meta-Models 75
6.1 MDE based Domain Specific Interface Metamodel Semantic Alignment Approach: An Overview 76
6.2 Possible Solution to Challenges of MDE based Semantic Mapping Approach: Extension of Interface Metamodels to Ontologies for Semantic Alignment 80
6.3 Strengths using Ontology based Approach: In contrast to MDE 80
Chapter 7 Extension of MDE based Modeling Approach to Ontologies for Evolution of Domain Unified Interface Ontology 82
7.2 Implementation of M3 Transformation Bridge 84
7.3 Platform-agnostic, Mediator Centric Approach towards Exploration of Semantic Synergies between Component Framework Interface Ontologies 88
7.4 Brief Overview of the Ontology Development Environment (ODE) 92
Part IV Finale 95
Chapter 8 Conclusion 96
Chapter 9 Future Work 98
Chapter 10 Appendix 100
References 111

List of Figures

Figure No.	Figure Label
Fig. 1	Overview of heterogeneous modes of communication between outside world and vehicle.
Fig. 2	Conceptual representation of interaction of heterogeneous application frameworks' APIs for Vehicle domain usecases.
Fig. 3	Interoperability between vehicle domain heterogeneous software platforms' applications.
Fig. 4	Overview of semantic overlapping between vehicle ECU software platforms' interface models.
Fig. 5	Abstract layered system model for Vehicle Domain Unified Interface Meta-metamodel.
Fig. 6	Role of SWC interfaces for Containerized vehicle application Clusters.
Fig. 7	WorkFlow representation of communication between heterogeneous, distributed applications in vehicle domain.
Fig. 8	Abstract specification of semantics of a generic SWC interface model.
Fig. 9	Overview of heterogeneous modes of communication between outside world and vehicle.
Fig. 10	An Overview model of Timing constraints on ordering of Events in a EventChain.
Fig. 11	An abstract specification of syntax of a generic SWC interface model.
Fig. 12	Abstract view of OMG specified four-layered metamodel architecture.
Fig. 13	Ecore representation of EMOF.
Fig. 14	An ecore metamodel representing AUTOSAR Adaptive Software Component interface level.
Fig. 15	An abstract model representation of OWL2 metamodel.
Fig. 16	Expanded view of an excerpt of OWL2 metamodel.
Fig. 17	OWL2 metamodel specification for AUTOSAR Adaptive Software Component interface model.
Fig. 18	Overview of partial mapping UML profile to ODM and extraction of ODM from OWL DL for DSL generation.
Fig. 19	Abstract representation of VFB Timing behaviour for AUTOSAR SWC framework
Fig. 20	Representation of Timing Description model describing event chain for a SWC interface.
Fig. 21	ParkA_1.0 and ParkA_2.0 event model.
Fig. 22	State machine model for ParkA_1.0 and ParkA_2.0 event-driven communication pattern.
Fig. 23	Overview of Callback queues and spins used for event chain for ROS2 framework .
Fig. 24	Overview of BPMN model from TwoUse Toolkit using OWLizer for metamodel translations.
Fig. 25	Different levels of Semantics.
Fig. 26	XML to WSML transformation using Adapter.

Fig. 27	Comparison of MDE and ontology based approaches for platform-specific model to model semantic mapping.
Fig. 28	Overview of Analysis level.
Fig. 29	Overview of heterogeneous modes of communication between outside world and vehicle for IoT software solutions.
Fig. 30	Symbolic representation of SeatHeating SWC.
Fig. 31	Multi-level modeling approach for evolution of abstract Meta IDL model for vehicle domain.
Fig. 32	Illustration of the case study SWC communication interface model using ARXML.
Fig. 33	Illustration of the case study SWC communication interface model using FCDL and FIDL.
Fig. 34	Illustration of the case study SWC communication interface model using MDL and SDL.
Fig. 35	Illustration of the case study SWC communication interface model using AIDL.
Fig. 36	Illustration of the case study SWC communication interface model using ARXML.
Fig. 37	Illustration of the case study SWC communication interface model using Protobuf.
Fig. 38	Illustration of the case study SWC communication interface model using Thrift IDL.
Fig. 39	Illustration of the case study SWC communication interface model using OpenAPI specification version 3.0.3.
Fig. 40	Overview of semantics for all operations using OpenAPI specification.
Fig. 41	Overview of cross-enterprise application framework communication for vehicle services.
Fig. 42	Work Flow Illustration of vehicle service SWC communication API model using OpenAPI specification (OAS).
Fig. 43	Representation of RPC communication semantics using OpenAPI specification (OAS).Representation of RPC communication semantics using OpenAPI specification (OAS).
Fig. 44	Abstraction of interface semantic traits for a SWC composition
Fig. 45	Illustration of abstraction of standardized four-layered modeling architecture for vehicle domain application SWC .
Fig. 46	Abstract representation of platform-independent, generic Component-Port-Connector (CPC) model for vehicle domain application SWC interface.
Fig. 47	Graphical model representation (G1) of AUTOSAR Adaptive framework SWC metamodels' constructs for interface.
Fig. 48	Graphical model representation (G2) of Franca framework SWC metamodels' constructs for interface.
Fig. 49	Graphical model representation (G3) of ROS2 framework SWC metamodels' constructs for interface.
Fig. 50	Graphical model representation (G4) of ROS2 framework SWC metamodels' constructs for interface.

Fig. 51	Graphical model representation (G5) of AUTOSAR Classic framework SWC metamodels' constructs for interface.
Fig. 52	Illustration of Design Approach to semantic alignment of cross-enterprise vehicle SWC frameworks' interface metamodels.
Fig. 53	Autonomous Maneurvering case study involving cross-enterprise vehicle application frameworks.
Fig. 54	Representation of the ecore Metametamodel (MOF) with standardized four layered modeling architecture.
Fig. 55	Overview of an ecore metamodel.
Fig. 56	Illustration of case study's platform-specific vehicle application SWC frameworks' interface ecore metamodels.
Fig. 57	Abstract representation of platform-agnostic, vehicle domain-specific generic SWC interface ecore metamodel.
Fig. 58	Abstract representation of semantic alignment of interface sources using MDE based ecore to ecore mapping approach.
Fig. 59	Reference domain-specific interface ecore metamodel (DM) representing polymorphic interface semantic traits.
Fig. 60	Representation of SWC frameworks' interface method classes and their derived semantic associated relationships.
Fig. 61	Overview of methodology concept for semantically ontology mapping and integration approach.
Fig. 62	Generic comparison between ecore and OWL2 conceptual metamodeling specification layers .
Fig. 63	Abstract layered representation of M3 Bridge approach to transform interface metamodel constructs from ecore to OWL2.
Fig. 64	Overview of exporting ecore model to XML Schema (XSD) using emf tools.
Fig. 65	Equivalent representation of an EPackage with XML schema.
Fig. 66	An example on exporting ecore metamodel constructs to equivalent XML schema using emf tools.
Fig. 67	Overview of schematic relation between XSD and RDF Schema (RDFS) .
Fig. 68	Illustration of schematic translation of XSD to equivalent RDF Schema using an example.
Fig. 69	Overview of translation of RDF Schema to OWL2 metamodel constructs using ontology framework supported tools.
Fig. 70	Workflow model for semantic alignment and integration of SWCs' interface semantic ontologies.
Fig. 71	Illustration of the case study SWC communication interface model using Thrift IDL.
Fig. 72	Semantic mapping of SWC frameworks' interface ontological metamodels using asserted axioms .
Fig. 73	Semantic alignment of SWC frameworks' interface ontologies using inferred axioms and reasoning.
Fig. 74	Exploration of semantic equivalence relationship between data-passing interface method calls using inferred axioms.

Fig. 75	Illustration of the conceptual global conceptual interface ontology schema for vehicle domain application SWC Frameworks.
Fig. 76	Overview of Ontology Development Enviornment (ODE) using Protégé.
Fig. 77	Overview of Workflow for Future Work.

List of Tables

TABLE I.	Contributions vs. Research Questions
TABLE II.	Overview of coverage of related works done in direction of current research
TABLE III.	Static semantic mapping table based on non-functional traits for vehicle frameworks' idls
TABLE IV.	Static semantic mapping table based on functional traits for vehicle frameworks' idls
TABLE V.	Static semantic mapping table based on communication protocol used for vehicle frameworks' idls
TABLE VII.	Semantic Mapping of Metamodel Constructs from Autosar Adaptive SWC CPC to Generic CPC Model
TABLE VIII.	Semantic Mapping of Metamodel Constructs from Franca SWC CPC to Generic CPC Model
TABLE IX.	Semantic Mapping of Metamodel Constructs from ROS2 framework SWC CPC to Generic CPC Model
TABLE X.	Semantic Mapping of Meta-model entities from Android framework SWC Construct to Generic CPC Model
TABLE XI.	Semantic Mapping of Meta-model entities from Autosar Classic framework SWC Construct to Generic CPC Model
TABLE XII.	Summarized Mapping Table for Semantic Mapping of Meta-model Constructs from Hetreogeneous framework SWC CPC to Generic CPC Model
TABLE XIII.	Summarized mapping Table for semantic mapping of meta-model entities from
TABLE XIV.	Semantic mapping Between XSD and RDF Schema Constructs
TABLE XV.	comparison of semantic alignment quality metrics for mde and ontology based conceptual metamodeling methods
TABLE XVI.	semantic mapping of interface functional traits between cross-domain idls using swc metamodels Specifications
TABLE XVII.	semantic mapping of functional traits between cross-domain idls using api code specifications

List of Abbreviations

SWC	Software Component
SOA	Service-Oriented Architecture
SW	Software
OWL DL	The Description Logics dialect of OWL
w.r.t	with respect to

Part I Introduction

With the increase in demand of software services in the automotive industry, automotive enterprises prefer to collaborate with other qualified cross domain knowledge partner enterprises such as Robotics, Telematics, Infotainment, etc. to provide complex automotive software services, such as autonomous driving, Over-The-Air Update, V2X (Vehicle-To-Communication), IoT (Internet of Things), etc. as also illustrated in Fig. 1and Fig. 2. The growth in cross-enterprise collaboration within the vehicle domain are due to the facts for example:

- firstly, the increase in requirement for integration with 3rd party and legacy components.
- secondly, the conformance to frequent new standards in vehicle domain [16].
- thirdly, the non-functional system requirements such as performance footprints, scalability, satisfiability, etc.
- lastly, the requirement of huge number of communicating processors for cross-domain communications between application components.

To deal with the mutual collaboration of services between vehicle domain cross-enterprise component frameworks, one possible option is to make each component implement several interfaces, which makes the software interface unnecessarily big. A second possible option may be to provide different implementations of a single component for each of the automotive development environments. Such a solution, however, will increase the development and test effort. There have been multiple frameworks of heterogeneous origin, covering several subdomains of the vehicle application domain and are often used to potentially collaborate in the design of complex vehicle applications or services. Possibility of several problems that could emerge due to the usage of these multiple frameworks' components in the vehicle domain at an application component level, are:

- *Composition of framework Control:* frameworks are often assumed to be in control of the application. When multiple such heterogeneous frameworks components coexist within a vehicle ECU, to accomplish the requirements for a complex vehicle service, synchronizing their functionality would be challenging. At the same time, it is also not possible for a single vehicle application component framework to cover the entire spectrum of vehicle services.
- *Overlap of framework functionality and Reduction in Reusability*: The opposite problem may also arise if different framework components provide overlap in their functionalities causing unnecessary overheads and hence reduction in reusability, substitutability and overall efficiency.

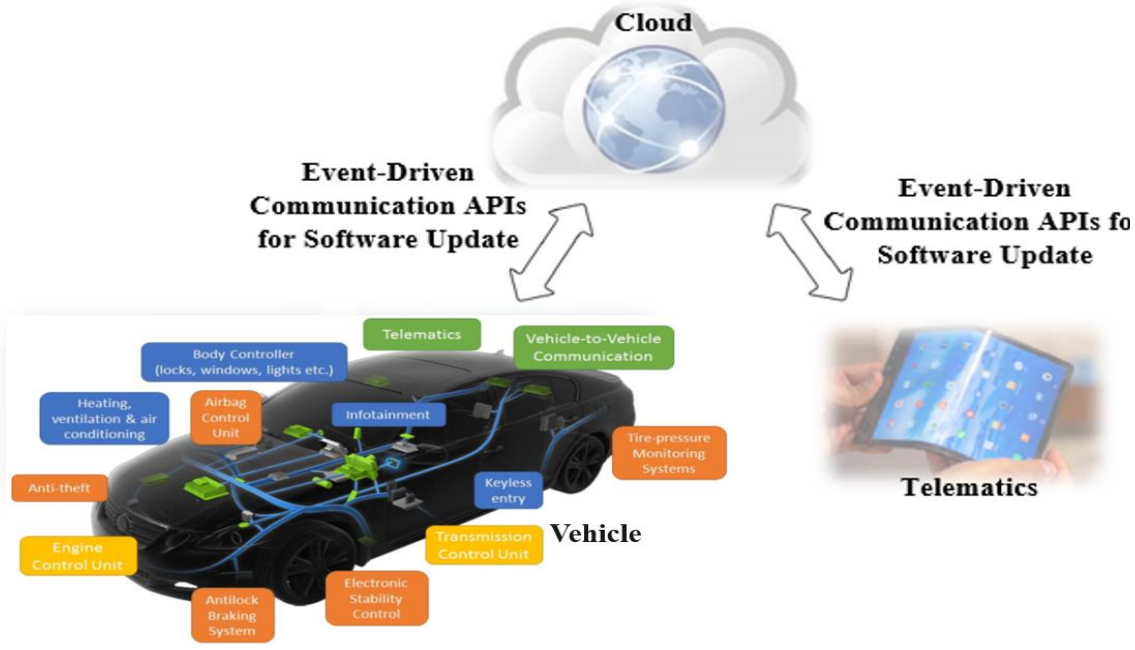


Fig. 1. Overview of heterogeneous modes of communication between outside world and vehicle.

The increasing demand on interoperable frameworks and solutions in the last five years is invoked by adopting the advancements of service-oriented architectures (SOA) and web services. It is particularly notable in the areas of

business-like automotive enterprises where there is a constant pressure on data and information exchange between the services, data resources, and applications distributed among a wide community of stakeholders of heterogeneous knowledge domains, also illustrated in Fig. 1. The key element that can be seen as a major criterion in reaching a high level of cross-enterprise collaboration between different services provider domains like Automotive, Telematics, Cloud, Robotics etc. is interoperability especially semantic interoperability between the applications or services' APIs. Interoperability in the field of information software systems stands for an ability of the seamless interoperation of the possibly heterogeneous services which may be provided and consumed by various independent actors in a networked environment, as conceptually represented in Fig. 2. In today's world, usually, any standalone framework in vehicle application domain cannot provide the complete spectrum of vehicle services, in such cases, to realize complex and novel vehicle services, automotive application software developers should instead focus on ease of interoperability with the application software developers from other functional knowledge domains such as telematics, Cloud, Robotics, etc.

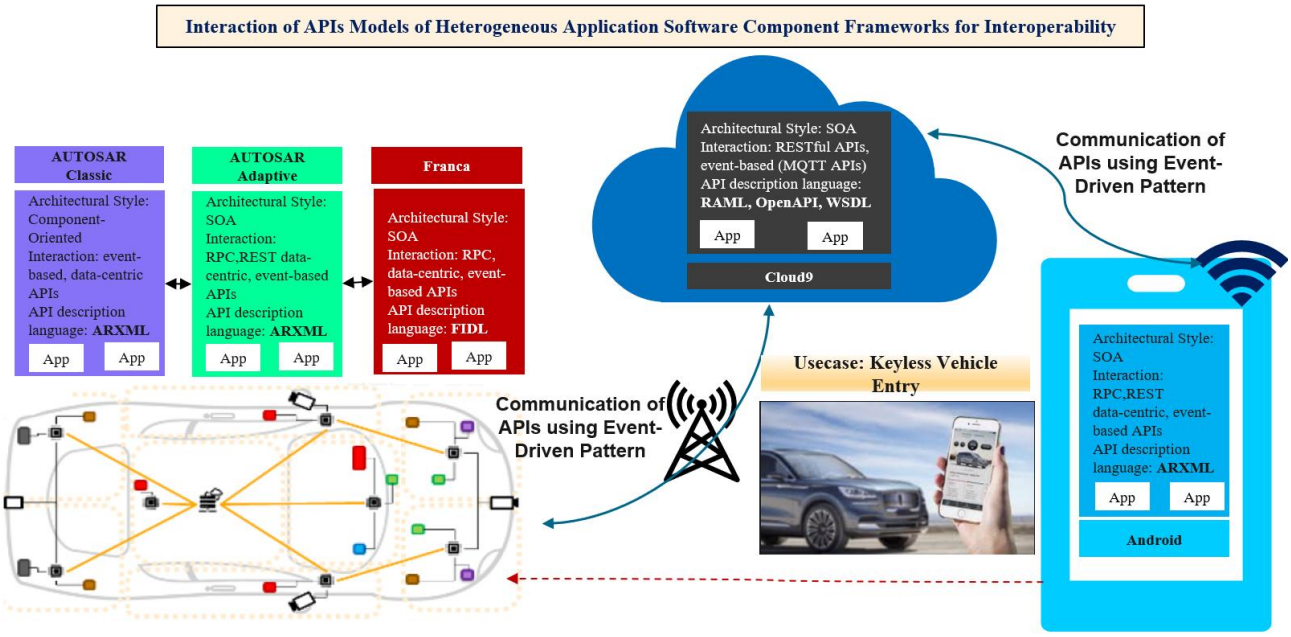


Fig. 2. Conceptual representation of interaction of heterogeneous application frameworks' APIs for Vehicle domain usecases.

Considering for example, a complex and novel usecase from the vehicle domain, such as *Keyless Vehicle Entry*, where the owner of a car wants to give the vehicle access to someone just by using his mobile phone, when the owner of the car is physically located far away from his car. Apart from good network connectivity, this complex vehicle domain usecase would also require good interoperability between the heterogeneous application frameworks' APIs. Therefore, as illustrated conceptually in Fig. 2, it has become necessary for the automotive members of this new ecosystem, to agree with other collaborating knowledge domain partner members in using a somewhat common service-based API or interface description format for the applications interoperability and for the realization of the vehicle domain complex usecase [6]. This will thereby ensure increase in the overall efficiency within the given new ecosystem scenario [6]. Additionally, for such complex and novel usecase, the automotive application software developers must also focus on reusability of software API modeling methods and technologies in a platform agnostic way based on the synergies in semantic interface traits of collaborating cross-enterprise application component frameworks.

From a modeling perspective, to align with the current research scope on the substantial role of semantic alignment of application software component frameworks' API models in achieving semantic interoperability, the Part I of the research work introduces the existing problem statement on semantic interoperability between software component frameworks with multiple emerging research questions and proposes possible solutions to the research questions through list of contributions and research goals. This Part also emphasizes on the usage of different conceptual metamodeling technologies for semantic mapping of vehicle domain SWC frameworks API models for interoperability and presents the relevant state-of the-art and the background knowledge on the same. Chapters in Part I also provides information on the layout of the research report along with the information on publications done so far to communicate the research work to the outside world.

Chapter 1 Research Goals and Contributions

Interoperability usually refers to the capability that depends on the understanding of interfaces. Interoperability is one of the major challenges to be addressed in achieving efficient software application cooperation, within and among enterprises. Over the past few years, there has been a wide spectrum of various Interface Description Languages (IDLs) proposed to describe application software components' interfaces within vehicle subsystems, as seen in Fig. 4. IDLs are special languages based on algorithmic notions that define those constituent elements of the SWC architecture, that specify connection of components with the connectors, and the rules of their correct composition. However, a source of discord is the level of support, a description of a specific component frameworks' interface description model provides, to read and understand the architecture of an application component for experts from other different sub-domains for mutual collaboration of services. This could be possibly since most component framework interface description models' specifications are not abstracted independent of middleware instead, they have bindings with platform-specific middleware communication protocols, which increases ambiguity, as seen in Fig. 3.

Semantic interoperability at application level depends on the degree of semantic alignment between application component frameworks' communication interfaces. Among the heterogeneous artifacts produced by multiple interface modelling languages in automotive domain, Model Driven Engineering (MDE) faces challenges for semantic alignment among multiple artifacts and formal semantic notations of each modeling language [6]. To ease semantic data heterogeneity caused due to the heterogeneous artifacts produced by various platform specific IDLs within the vehicle domain, it is essential to have semantic alignments based on domain interface semantic concepts. Additionally, it is also essential to focus on standardization of semantic interoperability process within vehicle domain, that means, it is time to focus on standardization of the semantic ways to access the vehicle services within the domain by using a unified, shared vocabulary to describe the interface traits of vehicle services among all the collaborating cross-enterprise component frameworks.

From a modeling perspective, semantic alignments between domain component framework interface models is however possible and can be made much easier, by shifting the focus from modelling language-centric to concept-centric [5]. That is, focusing on a component frameworks' interface semantic concepts independent of platform-specific modelling language implementation [1]. Ontology technology addresses the needs of semantic alignments between heterogeneous artifact by identifying, abstracting commonalities and checking for inconsistencies by using shared vocabulary of domain-specific terminologies. Exploring semantic synergies between component frameworks' interface models using ontology bridges the semantic gap between the domain component frameworks by increasing possibilities of correlation and reusing of few of the existing application SWCs' API models for further semantic integrations.

1.1 Research Questions

In the past few years, with the migration of most of the automotive software engineering sub-domains architectures to service-based architecture e.g., SOA (Service-Oriented Architecture), support for automotive complex and novel services requirements such as autonomous driving, vehicle-to-vehicle communication (V2V), etc. has increased. To support these service requirements, application component frameworks with heterogeneous API architectures and interface semantics must be integrated into collaborative systems of the future vehicle ECUs' (Electronic Control Units) High Performance Computing (HPC) software platforms, as seen in Fig. 4.

From a modeling perspective, it is usually observed that a vehicle application SWC's interface model has a commonality in fundamental semantics, despite using different syntactic representation, when modeling the same vehicle application interface in different frameworks using different IDLs. These further causes a decrease in efficiency and reusability of vehicle application SWCs. As a result, there can be an overwhelming number of ways on how to implement even a simple two-way communication using legacy or open-source platform specific framework tools. However, the logical functionality of a two-way communication like Client-Server, Sender-Receiver or event-driven communication design patterns might remain the same in much of the frameworks. In this context, exploring semantic commonalities in interface and interaction paradigms among SWCs of cross-domain frameworks to achieve interoperability among them, is the prime focus of this research contribution. Even with the commonality in semantics, the various application SWC frameworks' API models within the vehicle domain fail to interoperate with each other for a mutual agreement of services within a vehicle ECU sub-system, in the lack of a common unified representation template or vocabulary for service interoperability, as illustrated in Fig. 4.

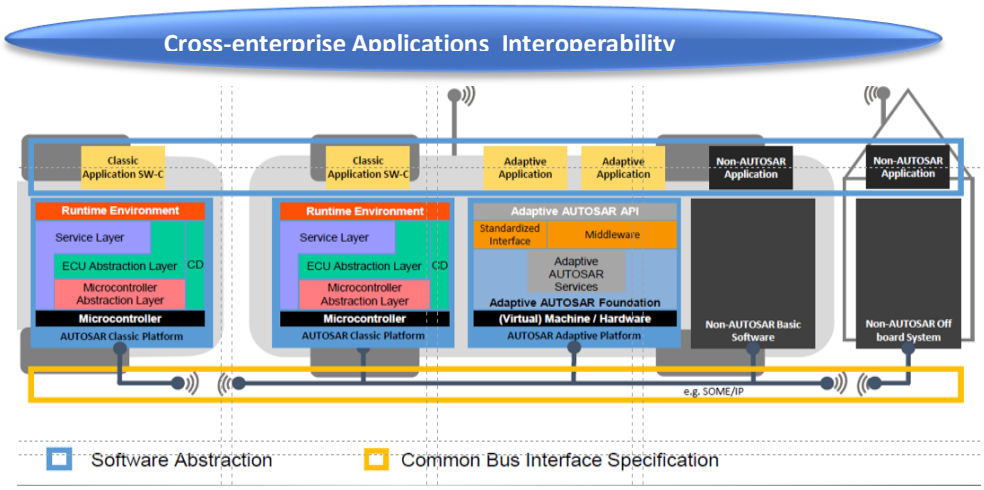


Fig. 3. Interoperability between vehicle domain heterogeneous software platforms' applications.

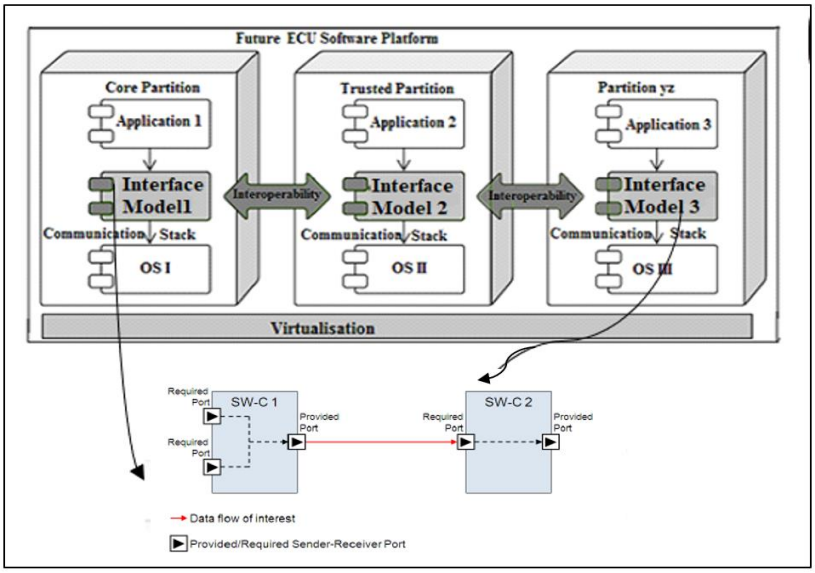


Fig. 4. Overview of semantic overlapping between vehicle ECU software platforms' interface models.

In the context of interoperability between heterogeneous application component frameworks' API models, the major research questions that has emerged and gained significant interest among the automotive software engineering communities is as follows:

What are the building blocks of an optimum design solution to semantic alignment of interface models of vehicle domain cross-enterprise software component frameworks for semantic interoperability? Could this design approach be implemented practically for vehicle domain case studies? Could the efficiency of such a design solution be estimated based on the evaluation of some semantic alignment metrics?

The above major Research Questions (RQ) could further lead to the following sub-questions:

RQ1: What are the basic functional and non-functional communication interfaces' semantic traits of vehicle component frameworks that are to be considered for communication between SWCs using the interfaces? Are there any possibilities of semantic overlapping between these interface traits of various vehicle component frameworks from functional perspective?

Why?

a) With the identification of vehicle application domain heterogeneous component frameworks basic interface semantic traits, possibilities of exploration of synergies among the interface semantic concepts of various vehicle component frameworks increases, thereby, increasing the possibility for semantic alignment of interface concepts for semantic interoperability.

b) Semantic analysis of heterogeneous component frameworks IDLs and generated codes within vehicle application domain provides a possibility to explore semantic synergies among interface traits used in various API specifications and represented as IDL generated code. This also provides possibilities to fill the semantical gap between the framework IDLs despite of syntactic differences.

How?

- Identification of fundamental interface semantic traits as perceived at Vehicle software components level externally.
- Semantic Analysis of component frameworks' IDLs within vehicle application domain using relevant case studies.
- Semantic Analysis of framework specific communication protocols associated with platform-specific component framework IDLs for middleware communication.

RQ2: What are the commonalities between the metamodel based domain specific conceptual modeling techniques employed by vehicle SWC frameworks in terms of interface spec? Can the interface metamodels be modeled in a (semi)formal way so that reasoning can be done about them?

Why?

By tracing the commonalities between MDE and Ontology based conceptual metamodeling methods using metamodeling languages like Ecore, OWL, etc. towards vehicle frameworks' interfaces semantic alignment, ensures further possibilities of amalgamation of these modeling methods in a complementary sense.

How?

- Analysis of existing conceptual metamodeling techniques of MDE and Ontology paradigms towards vehicle domain SWC frameworks' interface traits semantic alignments.
- Semantic analysis of constructs of MDE based metamodeling languages used for representation of interface metamodels.
- Semantic analysis of constructs of ontology-based metamodeling languages or specifications representing interface metamodels or schemas.

RQ3: Both MDE and ontologies are used to model concepts using different metamodeling methods. Can these metamodeling methods be amalgamated together to improve the interoperability on semantic level of vehicle SWCs' interfaces? Can the design approach to amalgamate MDE and ontology metamodeling methods be applicable practically to vehicle domain case studies?

Why?

a) Understanding and utilizing the strengths of ontology technologies to compensate the challenges in MDE like absence of domain conceptualization using shared vocabulary, automated reasoning, inferred axioms, dynamic classification and consistency checking which are otherwise absolute essential for leveraging the development of efficient and optimum solutions to semantic interoperability.
b) Applicability or Usability of applying MDE and Ontology metamodeling design approaches for evolution of vehicle domain Interface solution can be validated using quality metrics. Moreover, knowledge derived from such analysis of quality metrics are required for future implementations.

How?

- Improvement in design approaches towards evolution of a holistic, vehicle domain global interface solution can be achieved by mapping or extending of conceptual platform-specific component interface constructs of model-driven metamodels such as Ecore metamodels to equivalent constructs of ontology metamodels such as OWL2 metamodels.
- Validation of results of semantic alignments between component framework interfaces and semantic integration of these interfaces to a domain global interface ontology can be achieved by using evaluation of semantic alignment metrics for both ecore metamodeling and ontology metamodeling approaches. Alternatively, for the ontology metamodeling approaches, the validation of concept could also be achieved by extending reasoner with query engines like SPARQL**.**

1.2 Overview of Contributions

In context of semantic interoperability of component framework interface models, Model Driven Engineering (MDE) based approaches and ontology-based approaches can offer alternative solutions to address semantic interoperability within a domain. To confront this crucial requirement for semantic interoperability and to increase in effect the reuse of existing components through their interfaces, this research work compares MDE, and ontology paradigms approaches towards conceptual metamodeling and identify the commonalities and variations in these approaches. With identified commonalities, this work attempts to amalgamate MDE with ontology-based approaches in a value-added way towards the direction of evolution of a holistic or unified domain interface solution for automotive domain to tackle challenges due to semantic interoperability. Defining a unified domain global interface meta-metamodel would be able to standardize the semantic interoperability process model among the heterogeneous vehicle component frameworks at application level. That means, each service collaborating vehicle component framework uses a standard, unified domain vocabulary to describe the semantic traits of its interface model, so that the semantics of the APIs could be easily understandable by the other collaborating component frameworks, this further ensures increase in possibilities of identifying semantic commonalities among the collaborating frameworks and increase in co-relations among them much earlier in design phase.

This research work presents contributions of different natures, using a semi-automated approach to extend automotive domain heterogeneous component framework interface metamodels to ontologies using a transformation bridge in order to carry over the advantages from ontology technologies such as semantic interoperability to MDE based software modeling domain. Using the transformation bridge, the basic constructs of vehicle component frameworks interface metamodels described by ecore metamodeling language are mapped to corresponding constructs of OWL metamodel.

Proven from literature [1][3] seamless explicit reasoning and inference engines are the vital features of the ontology technology which provides benefits of semantic alignment and semantic integration. Taking this fact into account, as a next step, the semantic synergies between various automotive component framework interface metamodels represented as OWL ontologies are explored and validated using reasoner and evaluation of semantic alignment metrics. The approach to interfaces semantic synergies exploration is demonstrated using vehicle domain case studies and inferred interface semantic mapping axioms are verified using SPARQL queries [3]. The semi-automated approach is an approach towards evolution of unified domain interface software solution for automotive domain. This approach basically captures engineering information on vehicle software components interface semantic features, interface dynamic behaviors and communication or interaction. The approach to evolution of automotive unified domain interface software solution can be further broken-down into four abstract system levels, as illustrated in the Fig. 5.

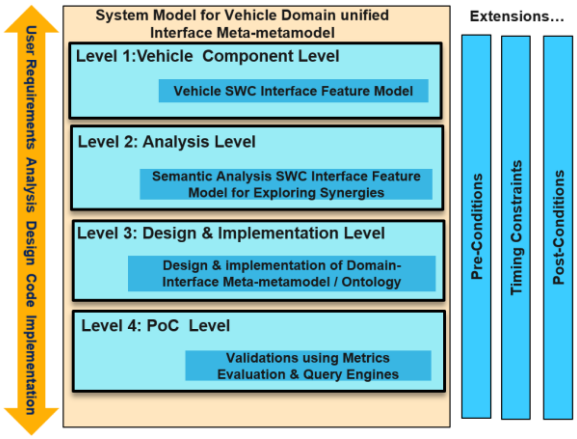


Fig. 5. Abstract layered system model for Vehicle Domain Unified Interface Meta-metamodel.

The four abstraction levels including the brief overview of contributions covered by the approach at each of these abstraction levels to evolve unified vehicle domain API model or solution includes basically:

Contribution at Level 1 (C1): This contribution identifies vehicle domain application SWC frameworks specific interface semantic traits that can be further considered as common knowledge base for semantic analysis and comparison of these frameworks' API specification models, architectural design patterns, IDLs, etc. This level also explains about the conceptual building blocks of an interface metamodel specified using MDE based ecore

and Ontology based OWL2 (Web Ontology Language) metamodeling languages. Brief overview of role of Ontology definition Model (ODM) to bridge the semantic gap between the ecore and OWL2 specified vehicle SWCs' interface metamodels is also covered in this level.

Contribution at Level 2 (C2): This contribution makes a survey of heterogeneous cross-enterprise SWC frameworks that are existing in the vehicle application domain and compares the semantic traits of these frameworks' API specification metamodels and IDLs. This comparison also explores semantic synergies among the interface semantic traits of the frameworks from a function point of view. Hence, with the exploration of semantic synergies among interface traits, this level ensures that cross-enterprise SWC frameworks can be correlated and the SWCs of these frameworks can be reused through their interfaces. This level also explores semantic synergies among the interface run-time behavior interface traits.

Contribution at Level 3 (C3): This contribution describes and compares the metamodeling techniques or approaches employed by ontology and MDE paradigms for alignment of semantically equivalent traits of interface metamodels of heterogeneous vehicle application component frameworks. Despite of few differences, focusing more on the similarities between the metamodeling approaches, the design approach proposes to amalgamate the MDE and ontology-based metamodeling techniques to utilize benefits of both the metamodeling techniques for semantic alignment and interoperability of various vehicle component frameworks' interface metamodel traits. This design approach considers the MDE and ontology-based metamodeling approaches as complementary and not as an alternative to each other, that means, challenges in MDE based metamodeling approach can be overcome with the strengths of ontology-based metamodeling approach and vice-versa. However, at the time of writing, implementation of the proposed design approach is still work-in-progress.

Contribution at Level 3 (C4): This contribution verifies qualitatively the success of metamodeling approaches or techniques used by MDE and ontology paradigms by evaluation of few quality metrics. In context of current scope and semantic interoperability between vehicle SWC frameworks' interface models, each of these quality metrics are defined to measure the richness of semantic similarity relations that could be explicitly expressed with the interface metamodels using MDE and ontology-based metamodeling techniques. These metrics are validated using relevant vehicle domain case studies. The quality metrics also provides quantitative information overview for selection of the suitable metamodeling approach or technique for semantic alignment and semantic interoperability between vehicle application component frameworks' interface models.

Each contribution at each abstraction level stated above can be related to research questions as seen in the TABLE I. below.

TABLE I. CONTRIBUTIONS VS. RESEARCH QUESTIONS

	Research Questions						**Contributions**			
	Q1	**Q2**	**Q3**	**Q4**	**Q5**		**C1**	**C2**	**C3**	**C4**
Chapters										
1										
2	✔						✔			
3	✔						✔			
4	✔						✔			
5		✔						✔		
6		✔						✔		
7		✔	✔					✔		
8			✔	✔					✔	
9				✔					✔	
10			✔	✔					✔	
11					✔					✔

Each of the abstraction levels as described above has a specific role to play in evolution of vehicle domain unified API model or software solution. From the vehicle level stating *what* the vehicle software components interface models should do through semantic analysis level, design and implementation levels using MDE and ontology paradigms that define, at various level of abstraction, *how* this is done [21]. The proposed abstraction levels and the contained elements provide a separation of concerns and an implicit style for using the modeling elements. The extensions include run-time interface behaviors such as timing constraints, pre-conditions, post-conditions, which are also part of interface semantics. The realization of extension pillars in Fig.4 refers to the core elements in all abstraction levels. The timing constraints extension pillar has been included within the current scope, as apart of contribution C2. Work on the other extension pillars are still under progress.

1.3 Report Structure

This section lay out the structure of this research report with a brief overview of each chapter. Information related to publications done w.r.t to the research work are included also as a part of this chapter.

Part-I: Introduction (including Vehicle Component Level)

Chapter 1: This chapter introduces the problem statement on semantic interoperability between vehicle component frameworks with multiple research questions with brief overview of contributions to find possible solutions to answer the research questions. This chapter also provides information on the layout of the research report in the context of the given research scope.

Chapter 2: The background knowledge which stands as a perquisite to understand the given research scope on semantic interoperability between heterogeneous application SWC frameworks API models in vehicle domain, is presented in this chapter. The fundamental concepts of SWC's interface metamodel, analysis of the run-time behavior associated with event-chain SWCs' API models and analysis of the concepts and technologies surrounding the modeling approaches for semantic mapping of SWC interface metamodels, based on MDE and ontology paradigms are presented in this chapter.

Chapter 3: This chapter presents the state-of-art towards semantic mapping and alignment of heterogeneous application SWC frameworks interface models. The state-of-art presents the brief overview of the relevant contributions from literature done so far in the direction of the current research scope along with the information on publications done to communicate the research work to the outside world.

Part-II: At Analysis Level

Chapter 4: This chapter focusses on the survey and comparison of the semantics of various SOA based vehicle applications frameworks SWCs' interface models using mappings and to explore the synergies in semantic mappings. This is to ensure domain experts understand the semantic synergies in API specifications at code generation level and decide which semantic synergies to be considered for designing a domain specific Meta Interface description language or Meta-IDL or Unified IDL at code level for automotive application domain.

Chapter 5: This chapter focuses on the metamodeling perspective of SWC interface metamodels and hence presents the semantic survey of interface metamodels of various cross-enterprise SWC frameworks within vehicle application domains represented as Component-Port-Connector (CPC) models. Each presented CPC model is designed considering the hierarchically classified three distinct levels: Semantic, Behavior and Composition.

Part-III: At Design Level & Implementation Level (Work-in-Progress)

Chapter 6 : This chapter focusses on finding semantic correspondences between Ecore as M2 level metamodel and Ontology Definition Metamodel (ODM) such as OWL2 (OWL version 2) metamodel, as both formalisms are fit for conceptual modeling. Therefore, towards semantic alignment of heterogeneous platform-specific vehicle application domain conceptual SWC frameworks' API models, this chapter presents an MDE based design approach towards semantic mapping of different frameworks' SWC interface Ecore metamodels using MDE based tooling support for EMFs.

Chapter 7: For domain-specific semantic alignments and integration methods without ontology, the solutions always have several drawbacks. Based on this fact, this chapter presents a semi-automated design approach to extend automotive domain heterogeneous component frameworks' interface ecore metamodels represented as schematic data models such as XSDs (XML Schemas) to ontologies schemas such as Resource Description

Framework Schema (RDFS) used for representing OWL metamodels. The design approach uses a transformation bridge by which the the basic constructs of component frameworks interface metamodels described by Ecore are mapped to corresponding constructs of OWL metamodels. The approach is illustrated using vehicle domain commonly used case studies. This chapter focusses on semantic alignment of vehicle component frameworks' interface metamodels when represented as OWL 2 ontologies and presents the design approach to semantically integrate the given frameworks' interface ontologies within the vehicle application domain for the evolution of a domain global interface ontology prototype as a meta-standard.

Part-IV: Finale

Chapter 8: Based on the comparative analysis of complementary interface metamodel semantic mapping approaches driven by MDE and ontology paradigms, this chapter draws conclusions based on results achieved with the proposed methodology and validation achieved at PoC level.

Chapter 9: As the work presented in this research report is still under progress, in that perspective this chapter presents an outlook of the on-going research work towards illustration of semantic interoperability between vehicle services of heterogeneous origin and profiles using a domain global interface ontology prototype.

Chapter 10: This chapter is about Appendix related to the research work presented in this contribution and it includes mapping tables between various vehicle SWC frameworks.

Chapter 2 Background

Vehicle domain businesses required their heterogeneous systems and applications to communicate in a transactional manner. The Extensible Markup Language was one of most successful solutions developed to provide business-to-business integration. XML became a means of transmitting unstructured, semi-structured, and even structured data between systems, enhancing the integration of applications and businesses. Unfortunately, XML-based solutions for applications and systems integration were not enough, since the data exchanged lacked an explicit description of its meaning. The integration of applications must also include a semantic integration. Semantic integration and interoperability are concerned with the use of explicit semantic descriptions to facilitate integration [82]. The word *Semantics* is the study of relations between the system of signs (such as words, phrases, and sentences) and their meanings. As can be seen by this definition, the objective of semantics is totally different from the objective of syntax [82]. Following the same definition, a SWC API's semantic expresses a software component in terms of its operations, inputs, outputs, and underlying types, defining functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising the interface. A good API makes it easier to develop a program by providing all the building blocks, which are then put together by the programmer.

2.1 Role of Interfaces in Interoperability of Vehicle Applications

Cars continue to turn into real cyber physical systems, just connecting to the internet and exchanging data with smartphones is the state of the art. Future cars will be connected to almost everything: Smart homes, roadside infrastructure, etc. Apparently, the architectures of todays' cars can be clustered into different domains for infotainment and connectivity, chassis, powertrain, etc. [6]. Collaboration between these domains is highly demanded to satisfy the novel automotive requirements [15][16] . However, for these collaborative environments to perform efficiently, interoperability stands as a major challenge. Nevertheless, to achieve a holistic data exchange between heterogeneous component framework interfaces, the semantics must be considered [6]. In addition, the notation used within the different semantics, understanding the notations without a general agreement, which makes the understanding between systems more complex. The purpose of any domain analysis is to select and define the domain of focus and to collect relevant domain information and integrate it into a coherent domain model. It is time to shift focus from the implementation of a certain modeling language towards the explicit semantic concepts covered by this language [1].

In the vehicle domain, perception of the role of interfaces in interoperability between future containerized vehicle application components, component frameworks with heterogeneous API architectures and semantics must be integrated into collaborative systems of the future ECU HPC software platforms to support automotive complex services, each application or application cluster running parallelly in containers. With the new era of automotive complex services requirements, multiple vehicle software services run parallel to one another on the same ECU HPC software platform. These services are broken down to microservices and packaged within containers based on application clusters. The application clusters in containers are expected to run parallel to one another.

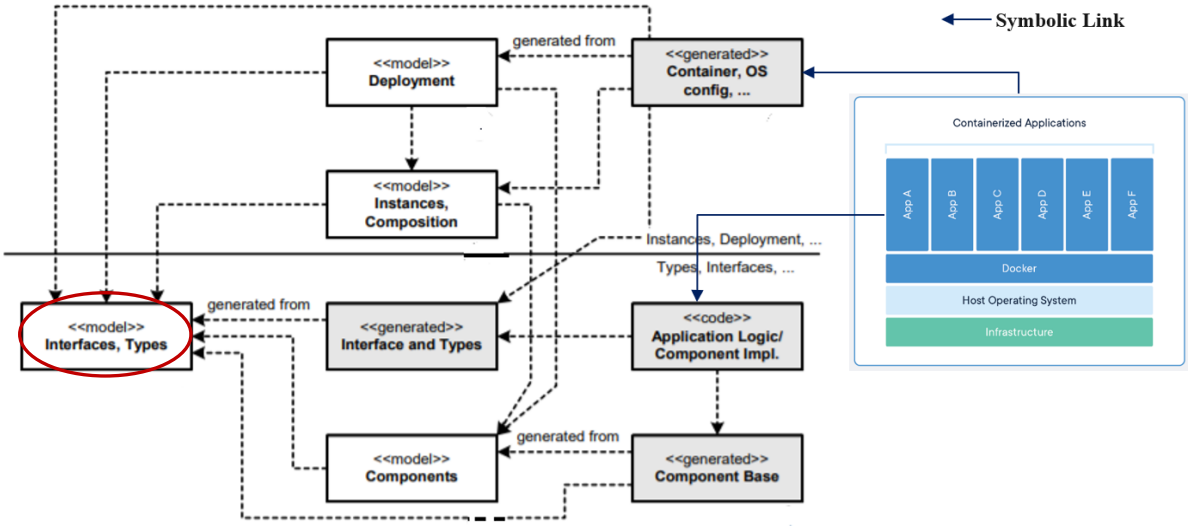


Fig. 6. Role of SWC interfaces for Containerized vehicle application Clusters.

Containers are an abstraction at the application layer that packages code and dependencies together. Multiple containers can run on the HPC software platform in parallel with one another, each running as isolated processes in user space. Each collaborating application cluster deploy their services to other collaborating partners co-existing in the same ECU HPC software platform using containerized microservices. For communication between containerized microservices, it is crucial to specify communicating port interfaces of application component instances. However, interoperability between containerized application clusters for services collaborations remains a challenge. In such scenarios, the interoperability majorly depends on the semantic commonalities between communicating port interfaces of application component instances or compositions, as also depicted by Fig. 6.

Interfaces define contracts between components, as well as between the container and the components it hosts. Since, an interface can be used by various components, hence, lifecycle callback interface of a component is used by the container to control the lifecycle of application component instances. This includes instantiating components, configuring instances, activating and passivating them over time, checking their state (running, standby, overload, error, …) or restarting one in case of severe problems. The propagation of events between application instances, synchronously or asynchronously can be supported by the container. For interoperability and communication between application instances or microservices in containers, time-based events can be triggered and notification interface in case interrupts occur can also be provided by the container. Timing constraints or pre/post conditions related to realization of interfaces can be checked by the container and errors can be reported.

Today, in the automotive industry the integration costs for enterprise apps co-operation are still extremely high, because of different business processes, data organization, application SWCs' interfaces that need to be reconciled, typically with great manual intervention. This problem has been addressed independently by MDA and ontology-based approaches. Applications in vehicle domain are implemented as multiple distributed components, and those components call each other's APIs for the complete application to function, as seen in . When developers design APIs for such applications, the solution characteristics they will typically prioritize are ease of programming for both the client and the server and efficiency of execution. Moreover, description of services in a language neutral manner is also vital for the widespread use of vehicle domain services. The containerized service providers must describe their containerized services and advertise them through their APIs in a standardized way like for example, using a universal service registry like commonly used in semantic Web domain, the Universal Description Discovery and Integration (UDDI) , an XML-based registry for business internet services, as seen in Fig. 7.

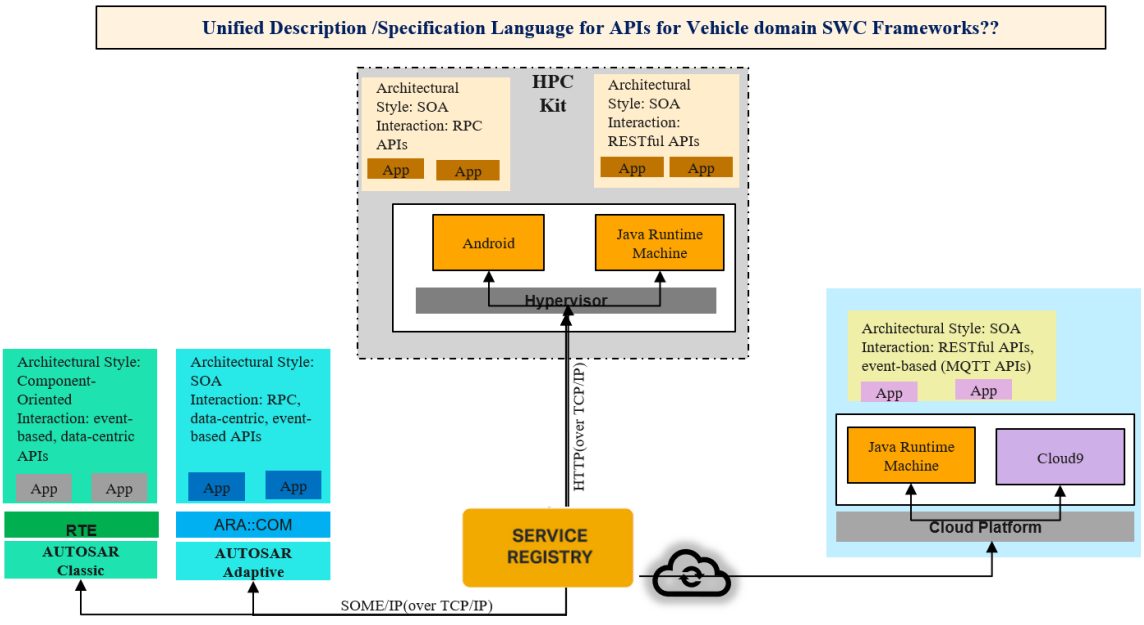


Fig. 7. WorkFlow representation of communication between heterogeneous, distributed applications in vehicle domain.

Legacy SWC frameworks like AUTOSAR Classic and Adaptive software exists (implemented with Remote Procedure Calls) that no longer fits its current needs or directions. This software is too valuable to abandon for vehicle application domain, and too difficult to change. One of the primary reasons that software is difficult to change is that basic assumptions are propagated through code from procedure to procedure.

In this scenario, One difficulty is the sheer variability of the interfaces and technologies that have to be integrated. Inspired by WSDL (Web Service Description Language) in Web Semantic domain, of using a common interface description language for describing the functionalities of Web services 'APIs, for widely distributed vehicle domain services, it can also be argued based on technical artifacts and emerging researches [84] to use a technology and platform agnostic, language neutral unified interface description template for describing vehicle domain containerized services'API specifications such as OpenAPI Specification (OAS).

The OAS defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. With other words, OpenAPI is a set of rules on how to describe with a human readable format the RESTful APIs. Once defined, a standardized OpenAPI method ought to have the same semantics when applied to any SWC frameworks' API specifications, though each API specifications determines for itself whether those semantics are implemented or allowed.

2.2 Fundamentals of Interfaces at Vehicle Software Component Level

From architectural abstraction perspective, a component can be considered as an opaque implementation of functionality which is a subject to third party composition. A component is conformant with a component model that is component models should prescribe how components interact with each other, and express architectural design. The ability to integrate components into assemblies, to reason about these assemblies, and to develop a market of components depends fundamentally on the notion of component interfaces constraints [40].

Technically, an interface of a component includes a set of named operations that can be invoked by clients. These set of named operations can be defined in a single or multiple service access points (or interfaces) of a component. However, it is important to note that other than definition, an interface offers no implementation of any of its operation [40]. Each operation's semantics is specified, and this specification plays a dual role as it serves both providers implementing the interface and consumer using the interface.

Most techniques for describing interfaces such as IDL are only concerned with the signature part, in which the operations provided by a component are defined, and thus fail to address the overall behavior of the component [44]. A more accurate specification of a component's behavior can be achieved through contract specification. In a group of software components, interface as contracts can also be used to specify the interactions between the components. Today most of the automotive and various other cross-enterprises' SOA based framework interfaces are specified as *Service Contracts*, allowing heterogeneous systems to communicate and interchange their services.

2.2.1 Overview of SWC Interfaces as Contracts

Like in any other engineering disciplines, from architectural abstraction perspective, an automotive domain software component model can be considered as an opaque implementation of functionality which is a subject to third party composition. Interface represented as a contract or *service contract* is a metaphor with connotations that are useful to CBSE (Component Based Software Engineering). For example, [40]:

- Contracts are between two or more parties.
- Parties (e.g. Service Consumer and Service Provider) often negotiate the details of a contract before becoming signatories.
- Contracts prescribe normative and measurable behaviors on all signatories.
- Contracts cannot be changed unless the changes are agreed to by all signatories. For example, changes due to versioning of service interface as *service contracts* must be freeze unless agreed to by all signatories.

The interface contract specification implicitly defines the component's behavior, the type of interaction between components to comply with architectural styles of a component framework for e.g., interface contract specification for *Event Subscription, Request Reply, Broadcast*, etc.

Interface semantics is defined as the actual meaning of interfaces of a component beyond their outer form, signature or appearance. Although syntactic specification of interfaces using contractual languages such as IDLs is in widespread use, but still it is widely acknowledged that semantic information about a component interfaces' operations is necessary to use the interface effectively and to reuse the component. Examples of such an interface semantic information may include the combination of parameter values an operation accepts, an operations' possible error codes, and constraints on the order in which operations are invoked. On the semantic level of an individual operation of an interface, there is particularly popular contractual specification method.

Semantically, the two sides of the interface as a contract can be captured by specifying *pre-* and *postconditions* for an operation [45]. The client or service consumer must establish the precondition before calling the operation, and the service provider can rely on the precondition being met whenever the operation is called. The provider must establish the postcondition before returning to the client and the client can rely on the postcondition being met whenever the call to the operation returns. An operation's *precondition* and *postconditions* will often depend on the state maintained by a software component [38], as depicted in the Fig. 8. For a given interface's state model an *invariant* also known as a predicate must always hold true for the correct realization of the interface, as also illustrated in Fig. 8.

When an interface's operation succeeds under specific conditions, these conditions are included in the contract's *precondition*, and then the contract's *postcondition* merely need to specify the outcome in those well- defined situations, this type of interface contract may be called as a *strong contract*. Whereas, if the contract's *precondition* for an interface operation is not able to filter out the invalid usecases, then the postcondition for such an interface operation will then specify the outcome of also invalid usecases, which needs to be filtered out as well, such contracts may be called as *weak contracts*. Nevertheless, p*re-* and p*ostconditions* functional aspects are not only the way to form contracts or specify all aspects of interface specification. In addition to *pre-* and *postcondition*, a contractual specification also includes *extra-functional requirements* like a so-called service level. The service levels cover guarantees regarding availability, mean time between failures, mean time to repair, throughput, latency, data safety for persistent state, capacity and so on. Failure to fulfill service level is treated as breaking of a contract. It is expected that this practice of including extra-functional specifications into a contract and monitoring them would become more widely used in the future [39].

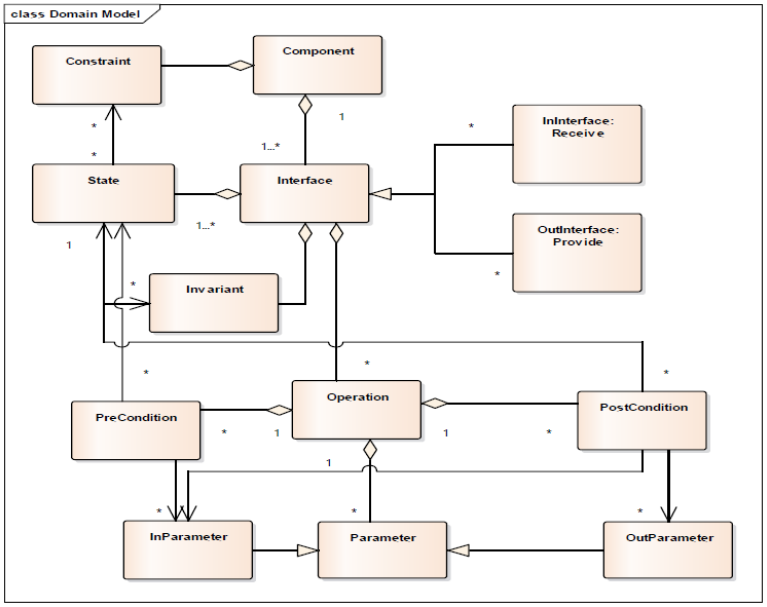


Fig. 8. Abstract specification of semantics of a generic SWC interface model.

Changes to a contract can take two forms. Either the interface or the specification is changed. If the interface is changed, this is referred to as syntactic change. If the specification is changed, this is called a semantic change. The problems caused due to contract changes are generally referred to as the fragile base class problem [39]. One possible approach to tackle the semantic changes is to insist on immutable interface specifications.

2.2.2 Vehicle Application SWC Interfaces Specifications at different Levels

Based on the state of the art [38][45][46] and common observations of a SWCM interface description , functionally interface specifications as contracts can be classified into four levels, they are namely Semantic, Syntactic, Behavioral and Interaction Level [45]:

2.2.2.1 Semantic Level

This level reinforces the syntactic level and concerns with the meaning of interface concepts or features often specified by the expectations or requirements. This level functionally includes *precondition*, *postcondition* and *invariants* that are required for implementation or realization of components' interfaces, however, in a group of

software components, the interface contract specification implicitly defines the component's behavior, the type of interaction between components to comply with specific architectural styles of a component framework. In general, the semantics for automotive service-based interface specification at the application software component level fundamentally includes [1][22][45].

- Separation of Interface Roles: The distinction between the consumer and service provider of a service interface.
- Distinction of interface types: operation-based (e.g. methods invocations), event-based (e.g. *Publish-Subscribe()* , *Broadcast()*, data-passing (e.g. *SenderReceiver()*), etc.
- Method invocation: Method calls with valid argument (or parameter) prototypes, e.g. *ClientServerOperation(), RequestReplyOperation(),* etc. including if any possible applicable post and pre-conditions and invariants.
- Event driven: This type of asynchronous interface specification includes *Callbacks (), Notifications*, etc.
- Data passing: One-way or unidirectional (*Fire-And-Forget ()*), Two-way or bidirectional (*SenderReceiver()*) function call semantics.
- Data Types: parameters' specifications like primitive, complex and user-defined data type specifications like int, float, string, Boolean, enum, array, vector, union, etc.
- The existence of some distinctive features appearing only for specific framework component models (such as special type of ports, optional operations)[45].

The overall interface specifications as a contract at semantic level also includes interface run-time behavioral constraints like timing constraints, etc. and interfaces interactions specifications.

2.2.2.2 Semantics of Component Interface Behavioral Constraints: Timing Constraints

Timing constraints are concerned with a SWC interface run-time behavior, hence, defined separately from the basic structural modeling elements of a platform specific IDL or SWC interface model. The fundamental concepts for describing timing constraints are that of *Events* and *Event Chains*. On the *analysis* level of system model abstraction, observable events can be identified, e.g., events that cause a reaction, i.e. a stimulus, and resulting observable event like a response, as illustrated in Fig. 9.

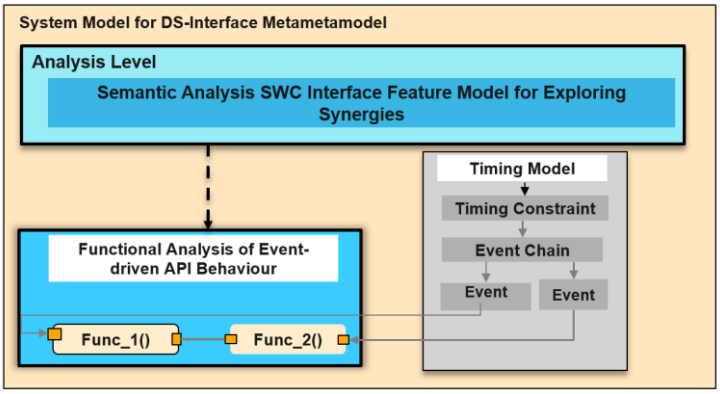


Fig. 9. Overview of heterogeneous modes of communication between outside world and vehicle.

Timing constraints represents dynamic behavior of interfaces as *events* or event *chains* based on timing analysis parameters. For example, timing constraints on the temporal ordering of events, etc.) [21].

The timing requirements can be imposed on *Event Chains*, for example, specifying that the time gap between the occurrence of a stimulus event and the occurrence of the expected response event, shall not exceed a specific amount of time, for example, an end-to-end delay from a sensor to an actuator. In addition, requirements regarding the synchrony of events can be expressed, stating that several events shall occur „simultaneously"in order to cause a reaction, or be considered as valid response of a system function. For example, in case of a passenger vehicle, its brake system shall apply the brakes simultaneously; or the exterior light system shall simultaneously turn on and off the rear and front turn signal indicators [37].

The timing constraints could be roughly grouped into restrictions such as the recurring delays between a pair of events, restrictions on the repetitions of a single event, and restrictions on the synchronicity of a set of events. Examples of timing constraints on the temporal ordering of events such as restrictions on the recurring delays between a pair of events like *AgeConstraint* and *ReactionConstraint*, restrictions on the repetitions of a single event, and restrictions on the synchronicity of a set of events like *Input* and *Output Synchronization Constraint*. All these timing constraints for automotive application software component interface models has been explicitly provided by TADL2 (of EAST-ADL) and are defined using UML class diagrams [37], as shown in Fig. 10.

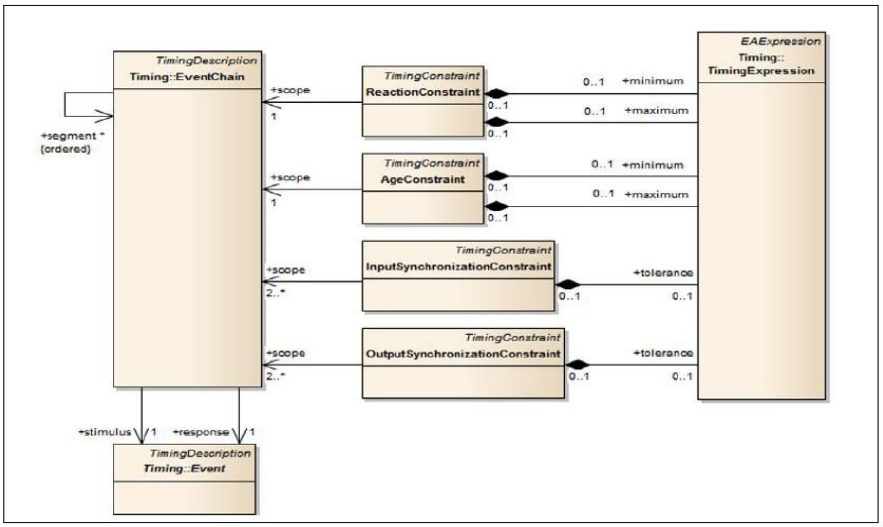


Fig. 10. An Overview model of Timing constraints on ordering of Events in a EventChain.

2.2.2.3 Semantics of Component Interface at Interaction Level/Composition Level

This level represents interactions between interface functionalities and behavior between multiple components as far as accessible through SWC ports [45][43] e.g. Synchronous, Asynchronous, etc. This level fundamentally includes following characteristics:

- Interaction Style using Software Connectors: Specification of software connections or connectors used for connection between SWC models. SW connectors can be realized in various types or forms of sw connections between SWCs depending on the used sw platform. For example, for interaction between SWCs interfaces, application framework for AUTOSAR Classic platform, uses sw connectors in two forms, namely *Assembly* sw connectors for interaction between atomic SWCs' interfaces and *Delegate* sw connectors for interaction between composite SWCs' interfaces [43].
- Type of Communication: which details mainly if the communication is *synchronous* (blocking) and/or *asynchronous* (non-blocking) between the components interface method calls.
- Method Mapping: Mapping of methods during service-based interface interactions, for e.g., Client methods can be mapped to one or more server methods and similarly server broadcast could be mapped to one or more client broadcasts.
- Parameter Mapping: parameters of server calls, client responses, client broadcasts) can be defined in a parameter mapping. A *parameter mapping* could consist of a mandatory default value and an optional arithmetic expression of input parameters. *Parameter mapping* could be considered as a part of interface method to *method mapping* at interaction level.
- Binding Type: describes the way SWCs may be linked together through the interfaces using communication protocols, for example, the Remote Procedure Calls (RPC) between SWCs' interfaces using communication protocol SOME/IP (Scalable service-oriented MiddlewarE over IP). *Binding Type* is an optional information for SWCs' interface specifications at interaction level.
- Interaction of component interfaces using sw connectors for example, for inter- or intra-ECU communications can be further realized in two subtypes:
    - The exogenous/endogenous sub-category describing whether the component model includes sw connectors as architectural elements.

- The hierarchical sub-category expressing the possibility of having a hierarchical composition of components (such as horizontal composition is an intrinsic part of all component models and thus it is implicitly assumed).

2.2.2.4 Syntactic Level

The basic syntax of interface model of a software component can be represented as seen in Fig. 11. This level describes the syntactic aspect and describes the signature of an framework interface generally using a specific language like IDL. This level relies mainly on an either static or dynamic, type checking technique.

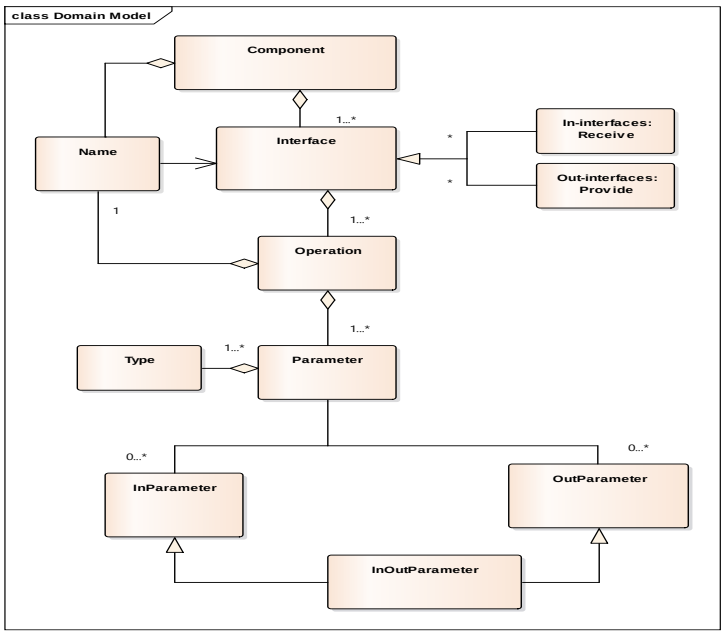


Fig. 11. An abstract specification of syntax of a generic SWC interface model.

This level fundamentally includes [45]:

- Interface Modelling Language Supported: The platform specific software modelling languages used to specify the interface for software component frameworks. Also known as IDLs.
- Method Signatures: As a part API specification, method invocation or operation structural elements or signatures such as names of methods, parameters specifications, etc. based on specific programming languages supported by different frameworks, are specified using syntactic level.

2.3 Interface Metamodel -The Conceptual Building Block

In the direction of semantic alignment of automotive component interface metamodels, it is essential to understand the fundamental constructs of MDE based metamodels and ontology based metamodels and then identify possible commonalities and variations between the metamodeling constructs of both the paradigms. To accomplish this goal, this subsection focusses on the basic literature of MDE and ontology technology paradigms in context of conceptual modelling especially metamodels. Metamodeling is the act of describing the model of a modelling language using another language, namely metamodeling language [3].

What are Metamodels?

Metamodeling is the act of describing the model of a modeling language using another language, namely metamodeling language [2]. If each metamodel is expressed using a modeling language, this could lead to an infinite hierarchy of meta-meta models. A common solution to this problem is to let a modeling language define itself at a certain level in the hierarchy as per OMG standardization, depicted by Fig. 12 [2]. MDA by OMG addresses *four layered metamodel architecture* namely, M0, M1, M2 and M3. M0-layer represents real world objects. M1- layer defines models based on simplification and abstraction of M0 layer. Models at M1 layer are defined using concepts which are described by metamodels of M2 layer. Metamodels of M2 layer are further described by concepts defined by meta-metamodels or Meta-Object Facility (MOF) of M3 layer [47]. The MOF is OMGs' standard specification for defining metamodels.

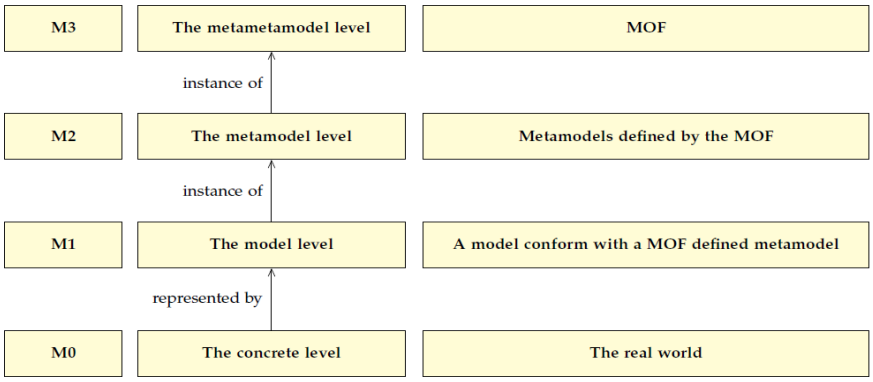


Fig. 12. Abstract view of OMG specified four-layered metamodel architecture.

2.3.1 MDE based Ecore Metamodel

The MOF provides a language to define abstract syntax for modelling language [24]. In general, the version 2.0 of MOF provides two types of metamodel definition, namely, Essential MOF (EMOF) and Complete MOF (CMOF). EMOF prefers simplicity of implementation before expressiveness. On the other hand, CMOF is comparatively more expressive but more complex to implement [3]. According to EMF literature, OMG (Object Management Group) addresses meta-metamodeling in Ecore as implementation of EMOF in M3 layer of four-layered modelling architecture [3][2]. Fig. 14 illustrates a simplified version of Ecore meta-metamodel [3]. A metamodel described by Ecore consists of mainly Packages which can be nested and contains set of Typed Elements. Packages, Typed Elements and Types are Named Elements. Types has two subclasses: Class and Datatype. A Class can be abstract or concrete. A Class can have Property. A Property is a Multiplicity Element and Typed Element.

Ecore is in its own metamodel and provides four basic metamodeling constructs when used with Eclipse Modeling Framework: (1) *EClass* is used for representing a modeled class. (2) *EAttribute* is used for representing a modeled attribute. Each Attribute have a name and a type. (3) *EReference* is used association relationship between classes. (4) *EDataType* is used for representing the attributes types. There are few meta-classes described by Ecore such as *EFactory*, *EOperation* and *EParameter* which do not represent concepts of modelling language, hence, is not considered in the current scope [3][4][5].

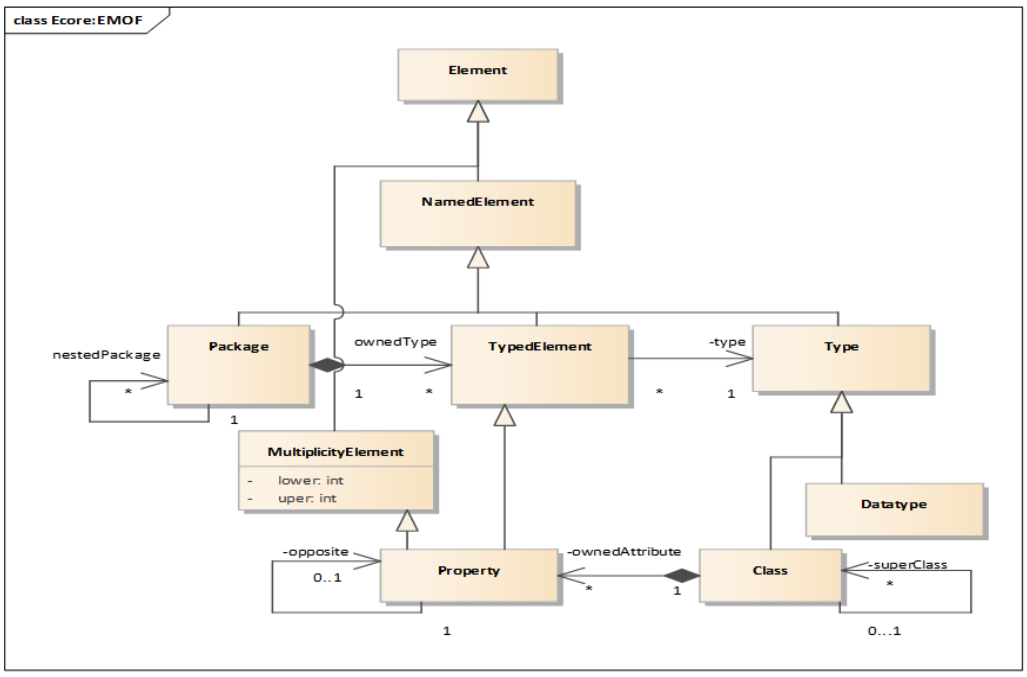


Fig. 13. Ecore representation of EMOF.

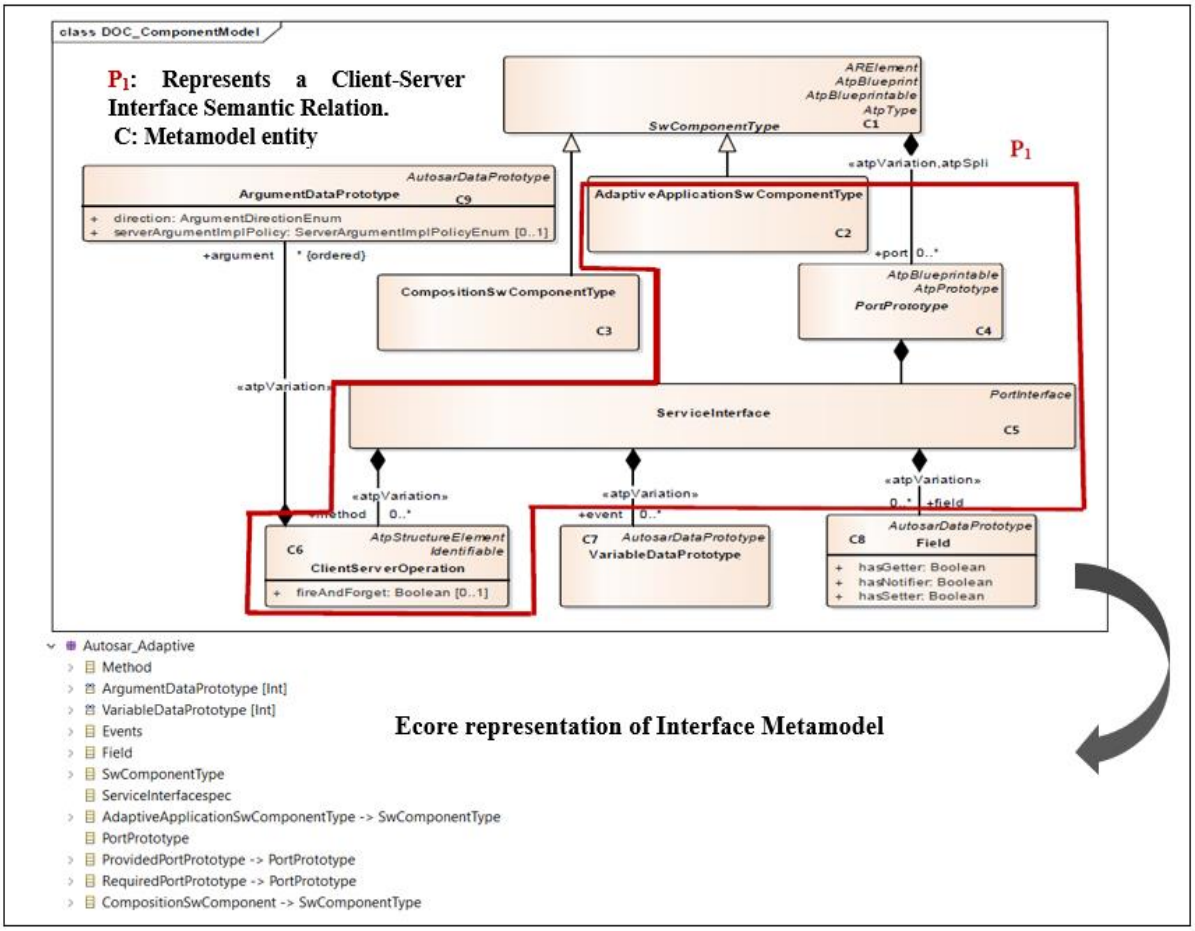


Fig. 14. An ecore metamodel representing AUTOSAR Adaptive Software Component interface level.

Fehler! Verweisquelle konnte nicht gefunden werden. illustrates a typical example of ecore metamodel (M2 l evel) describing an automotive domain platform-specific component framework namely, AUTOSAR (AR) Adaptive component framework communication interface model.

2.3.2 Ontology based OWL2 Metamodel

The OWL2 metamodel which is also part of the Ontology Definition Metamodel (ODM) specification defined in the OMG standard, is implemented by extending the RDFS (Resource Description Framework Schema) metamodel. Fig. 15 illustrates the class hierarchy present in OWL. An *OWLClass* is a kind of RDFS Class, like an OWL Property is kind of or inherited from *RDFProperty*. *OWLProperty* can be distinguished based on functional, data and object resources. With it we can define cardinalities on properties, define classes with set operators like union, intersection, complement, enumerated and OWL restriction [24].

OMG specifies RDF concepts and RDFS metamodels complies to MOF 2.0 version and are comparable to meta-metamodels of M3 layer of four-layered modelling architecture [3][2]. RDFS serves as a meta-language that defines itself and OWL. RDF classes are equivalent to MOF classes. RDF properties are represented either by MOF classes or associations, as appropriate [11] [7]. RDF properties are first-class entities with unique identifiers. An RDF property can be a sub-property of another RDF property, as can be seen in Fig. 15. MOF associations, on the other hand, are not first-class entities.

Although ecore and OWL metamodel have multiple semantic correspondence, however, when defining a common superclass for two subclasses to denote that all instances of the subclasses are also instances of the superclass. This specific intention would be equally satisfied in both ecore and OWL metamodel [5]. However, in ecore this also means that instances of either subclass can be instance of one of the subclasses only, whereas individuals in OWL could actually belong to both subclasses [1][5].

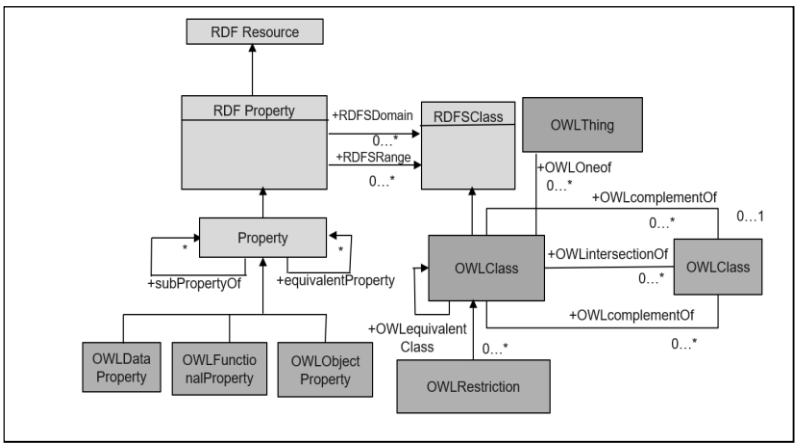
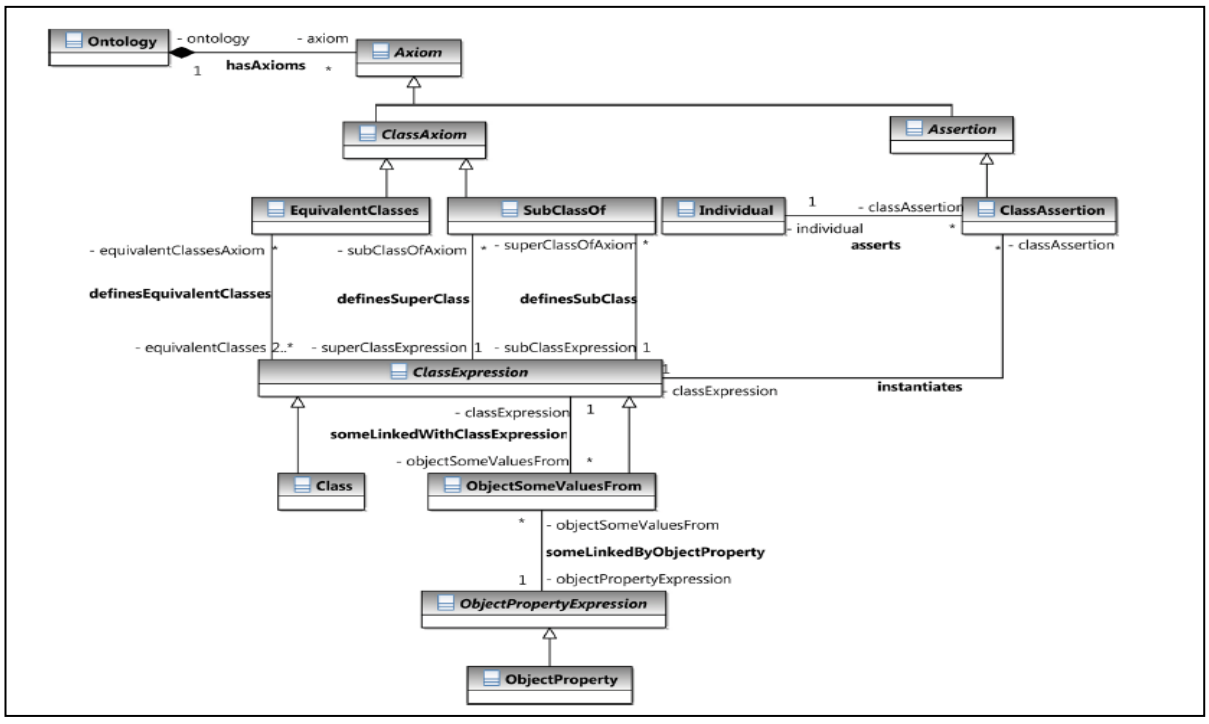


Fig. 15. An abstract model representation of OWL2 metamodel.

In contrast to EMOF, in RDFS, a MOF can be a sub-association of another MOF association [3][2][24]. The RDF concepts describes the concept of an RDF property as a relation between subject resources and object resources. All things described by RDF are called *RDFResource* [23]. All the other classes are subclasses of this class. Every RDF property is associated with a set of instances, called the property extension. Instances of RDF properties are pairs of RDF resources [24][47].

Fig. 16 depicts the essential features of OWL2 metamodel [9]. Each OWL2 ontology consists of a set of axioms. Class axioms for example are the *Equivalent-Classes* axioms or the *SubClassOf* axioms. The *EquivalentClasses* axiom is used to describe two or more class expressions as equivalent, whereas the *SubClassOf* axiom defines exactly one class expression to be the subclass of another. A possible class expression for example is the *ObjectSomeValuesFrom* expression which describes those individuals which are connected via a given object property to at least an individual of the given class expression. Beside class descriptions, individuals are part of an ontology [9]. Class assertions are *axioms* that are used to assert individuals having as type the given class expression.



.

Fig. 16. Expanded view of an excerpt of OWL2 metamodel.

Fig. 17 illustrates a typical example of the class hierarchy present in OWL2 meta-model (M2) specification or ontology for describing an automotive domain platform-specific component. framework, namely, AUTOSAR (AR) Adaptive component framework communication interface model.

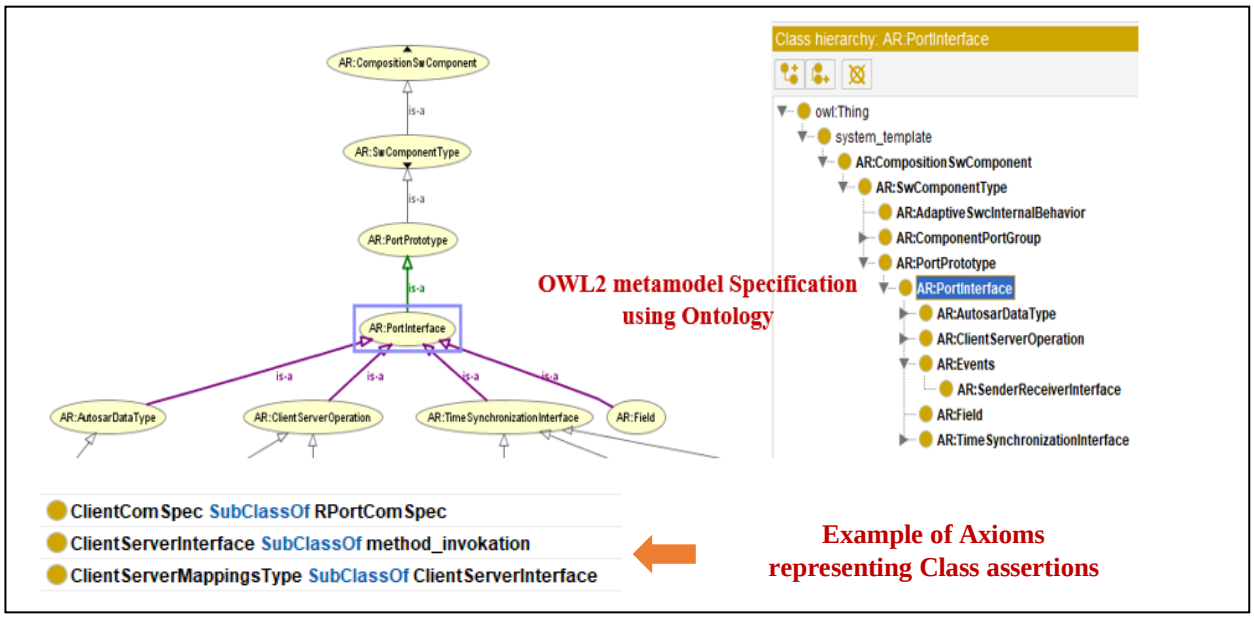


Fig. 17. OWL2 metamodel specification for AUTOSAR Adaptive Software Component interface model.

2.3.3 Ontology Definition Metamodel (ODM)

The ODM is applicable to knowledge representation, conceptual modeling, formal taxonomy development and ontology definition, and enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF [24]. ODM-based ontologies can be used to support [23]:

- interchange of knowledge among heterogeneous computer systems.
- representation of knowledge in ontologies and knowledge bases.
- specification of expressions that are the input to or output from inference engines.

The ODM specification offers several benefits including knowledge basis for conceptual mapping of metamodels from ecore to OWL2. The need for a dedicated ontology modeling language stems from the observation that an ontology cannot be sufficiently represented in UML [24]. Both representations share a set of core functionalities such as the ability to define classes, class relationships, and relationship cardinalities. Despite this overlap, there are many features which can only be expressed in an ontology language. Examples for this disjoints are transitive and symmetric properties in OWL or methods in UML. As per the OMG standard the key requirement for ODM is [23][24]:

- An ODM must be grounded in the Meta Object Facility2(MOF2).
- Bidirectional partial mappings between an ODM and EMF profile such as ecore , UML ,etc. defining visual notation for the same ontologies should be possible to be established as illustrated in Fig. 18.
- From the ODM, one must be able to generate an ontology representation in a language such as OWL DL, also seen in Fig. 18.
- An XMI serialization and exchange syntax for ODM must be provided. This XMI format allows exchanging an ODM metamodel between tools.

In the current scope of research contribution, the focus is more towards accomplishment of second and third requirements for ODM as the other two requirements directly follow from a good ODM. An ODM can also be extracted from an OWL DL. Ontology-Driven Software Development provides a quick and simplified description of a DSL (Domain-Specific Language), abstracting languages' technically details, while highlighting key terminology and specifics. Once an ontology is built, it is a simple process to generate the languages' metamodel and establish relationships among the related concepts. This languages' metamodel that could be generated from OWL DL using language mappings is an ODM [23]. The ODM specification offers several benefits to potential users [23], including:

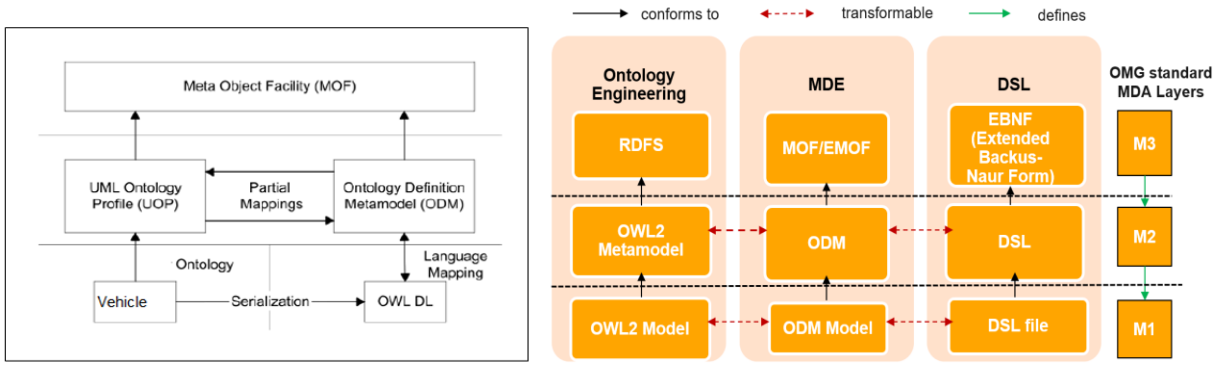


Fig. 18. Overview of partial mapping UML profile to ODM and extraction of ODM from OWL DL for DSL generation.

- Options in the level of expressivity, complexity, and form available for designing and implementing conceptual models, ranging from familiar EMF profiles like UML, ecore, etc. to formal ontologies represented in description logics or first order logic.
- Grounding in formal logic, through standards-based, model-theoretic semantics for the knowledge representation languages supported, enough to enable reasoning engines to understand, validate, and apply ontologies developed using the ODM.
- Profiles and mappings enough to support not only the exchange of models developed independently in various formalisms but to enable consistency checking and validation in ways that have not been feasible to date.
- The basis for a family of specifications that marry MDA and Semantic Web technologies to support semantic web services, ontology and policy-based communications and interoperability, and declarative, policy-based applications in general.

ODM specifications is applicable to knowledge representation, conceptual modeling, formal taxonomy development and ontology definition, and enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF [24][23].

2.4 Event Chain Timing Behavior of Software Components' Interface Models

As observed from Fig. 1, in today's era for the realization of most of the complex and novel vehicle functionalities or services we prefer event driven communication pattern when communicating with distributed, heterogeneous origin application component frameworks' interfaces for service collaborations and exchange of data. Therefore, within this given context, among all other existing run-time behavioral constraints for SWC frameworks' interfaces, it is most important to analyze the timing models associated with each of the service collaborating SWC frameworks that supports the event-driven API communication pattern.

The end-to-end timing model of SWC frameworks consists of the information containing timing properties, requirements, dependencies, control and data flows concerning all events, messages and event chains in the system. Conceptually, timing information can be divided into timing requirements of SWC frameworks and timing properties, where the actual timing properties of solution must satisfy the specified timing requirements [37][22]. Based on this information, the timing analysis can predict the execution behavior of the system with respect to end-to-end timing. Most existing approaches for component-based vehicular distributed embedded systems support the representation of such timing models at an abstraction level that is close to their implementation. Representation of the timing model at the higher abstraction levels is challenging mainly because not all timing information is available at the higher levels. Hence, the analysis results may not represent accurate timing behavior of an overall vehicle sub-system. The TADL2 (Timing Augmented Description Language version 2.0) developed by the TIMMO project, provides the only viable formal method for modeling of timing information using timing constraints at various abstraction levels in the vehicle domain. This is evident from the fact that TADL2 has recently provided the timing model to the EAST-ADL language and AUTOSAR [7]. EAST-ADL is an architecture description language in the automotive domain [22][37].

Most of the vehicular functions are developed as distributed embedded systems with real-time requirements specified on them. This means that the providers of the systems are required to ensure that logically correct actions

are taken by the systems at times that are appropriate to their environment (i.e., the timing requirements are satisfied). In the context of current scope, the timing behavior of the interface data-passing method calls, or *events* can be determined by calculating its response time. The response time of a task or a message is defined as the amount of time elapsed between its activation and completion or reception respectively. Often, vehicular embedded systems are modeled with task chains.

A task chain consists of several tasks that are in a sequence and have one common ancestor. At vehicle software component interface level, a task chain can be considered as *Event Chains*, where each Event Chain consists of number of *Events*. Each task may receive an activation trigger, a data or both from its predecessor. Any two neighboring tasks in a chain may reside on two different nodes, while the nodes communicate with each other via network work messages. In this case, the messages are part of the task chain. The timing behavior of the task chain is determined by calculating its end-to-end response time and/or delays. The end-to-end response time of a task chain is defined as the amount of time elapsed between the arrival of an event at the first task and the production of the response by the last task in the chain. If the tasks within a chain are activated by independent sources (e.g., clocks), then different types of end-to-end delays are also calculated to determine the timing behavior of the chain.

Event chains specify a causal relationship between events and their temporal occurrences. The notion of event chain enables one to specify the relationship between two events, for example when an event A occurs then the event B occurs, or in other words, the event B occurs if and only if the event A occurred before. In the context of an event chain the event A plays the role of the stimulus and the event B the role of the response. Event chains can be composed of further existing event chains and therefore can be decomposed into event chain segments.

Like event triggering constraints impose timing constraints on events and their occurrences; the *latency* and synchronization timing constraints impose constraints on event chains. In the former case, a constraint is used to specify a *reaction* and *age*, for example if a stimulus event occurs then the corresponding response event shall occur not later than a given amount of time. And in the latter case, the constraint is used to specify that stimuli or response events must occur within a given time interval. The next subsection illustrates more artifacts on the run-time behaviors of commonly used vehicle application SWC frameworks interfaces w.r.t. to event chains.

2.4.1 AUTOSAR SWC Framework Event Chain Timing Analysis: VFB Timing View

AUTOSAR timing models consist of timing descriptions, expressed by events and event chains, and timing constraints that are imposed on these events and event chains. The AUTOSAR *Specification of Timing Extensions* defines a set of predefined event types related to one of the AUTOSAR views: Virtual Function Bus VFB View (*VFB Timing*), Software Component View (*SWC Timing*), System View (*System Timing*), Basic Software Module View (*BSWM Timing*), and ECU View (*ECU Timing*) [43]. In particular, one uses these events to specify the reading and writing of data from and to specific ports of software components, calling of services and receiving their responses (*VFB Timing*); sending and receiving data via networks and through communication stacks (System Timing); activating, starting an terminating executable entities (*SWC Timing*); and last but not least calling basic software services and receiving their responses (*ECU Timing and Basic SW Module Timing*).

As an example of *VFB Timing* , consider the timing behavior illustrated in Fig. 19. From the point in time, where the value in is received by the software component named as SWC in Fig. 19 [43], until the point in time, where the newly calculated value out is sent, there would be maximum latency of 2 ms. This would be attached to the timing description that refers to an *AtomicSwComponentType* SWC.

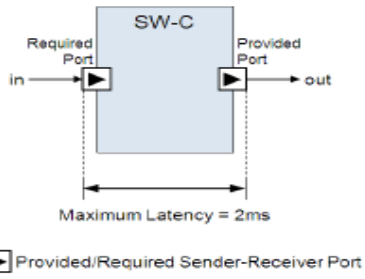


Fig. 19. Abstract representation of VFB Timing behaviour for AUTOSAR SWC framework

In view of current scope, we consider the TADL2 definition of *Timing Description model* for *VFB Timing* view for an AUTOSAR Classic frameworks' SWC's communication interface, as also depicted by Fig. 20[37][22].

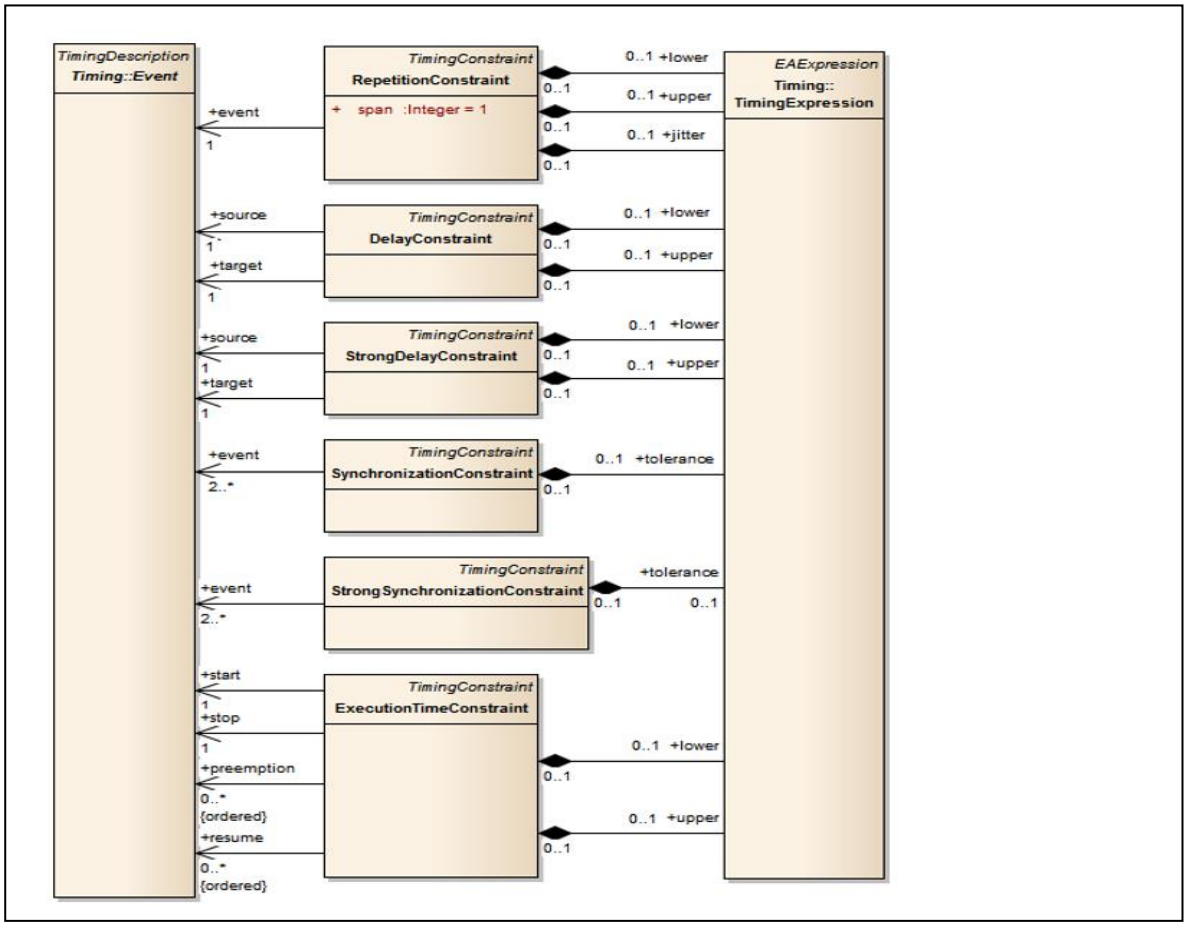


Fig. 20. Representation of Timing Description model describing event chain for a SWC interface.

An *ExecutionTimeConstraint* is used to specify the execution time of the referenced *ExecutableEntity* in the referenced component. A minimum and maximum limit for execution time can be defined. *RepetitionConstraint* means repetition of the event pattern for example, burst, concrete, and arbitrary pattern.

2.4.2 Franca SWC Framework Event Chain Timing Analysis

The syntactical interfaces are defined using the Franca interface description language (IDL) [FRA] which is the current standard for specifying GENIVI compliant infotainment interfaces. The syntax model includes information about the interface name, the interface version, methods, broadcasts, parameters and type definitions.

The event model extends the syntax model with the definition of signal events. A signal event is used in order to describe an occurrence of a method call, method response, or broadcast with a concrete parameter constraint. The signal event can be specified with the constraint attribute which defines a logical expression using the parameters of the referenced method. The call event can be constrained using the input parameters of the method and the response- and broadcast events can be constrained using the output parameters of the method or broadcast.

For the *Event Chain* timing analysis in Franca Framework, let us consider a scenario, say, a sensor vehicle SWC *ParkA* exists in two versions, *ParkA_1.0* and *ParkA_2.0*. Both the component versions call methods for SWC *startup* and *shutdown*. Also, *ParkA_1.0* defines a broadcast (started) for signaling the ready state, and one broadcast (sensorValues) for sending sensor values. *ParkA_2.0* defines three methods. Two for startup and shutdown and one for receiving of sensor values from *ParkA_1.0*, as illustrated in Fig. 21 [68].

```
import "../franca/ParkA1.fidl"

Interface Parka {
  CallEvent start {
    methodRef startup
  }

  ResponseEvent start{
    methodRef startup
  }

  CallEvent shutdown{
    methodRef shutdown
  }
  ResponseEvent shutdown{
    methodRef shutdown
  }
  BroadcastEvent sensorValues{
    methodRef sensorValues
  }
  BroadcastEvent started{
    methodRef started
  }
}
```

```
import "../franca/ParkA2.fidl"

Interface Parka {
  CallEvent connect {
    methodRef connect
    constraint {retry == false}
  }
  CallEvent reconnect {
    methodRef connect
    constraint {retry == true}
  }
  ResponseEvent connectOK{
    methodRef connect
    constraint {(result == OK)
      && (connectionID != -1)
    }
  }
  ResponseEvent connectERROR{
    methodRef connect
    constraint {(result == ERROR)
      && (connectionID ==-1)
    }
  }
  CallEvent disconnect{
    methodRef disconnect
  }
  ResponseEvent disconnect{
    methodRef disconnect
  }
  CallEvent getSAS{
    methodRef getSensorsAndStatus
  }
  ResponseEvent getSAS{
    methodRef getSensorsAndStatus
  }
}
```

Fig. 21. ParkA_1.0 and ParkA_2.0 event model.

For mapping of client-server interface method calls and parameters between *ParkA_1.0* client interface to *ParkA_2.0* server interface, timing description must be specified by setting timeout parameter for connection establishment to a specific value. Such a timing analysis or behavioral analysis of interface can be better explained using state machine diagrams, as illustrated in Fig. 22[68]. The dynamic adapter in Fig., maps a *ParkA_1.0* client interface to a *ParkA_2.0* server interface. The *active flag* is only shown if it is set to true. The *Max*-argument is only visualized if the timeout is set for a specific state. The *Max* attribute can be used to define the maximum active time of a state. If the *Max*-attribute of a state is set to some value, the state machine will start a timer while entering the state. Once the timer elapses a timeout signal will be generated which may be used as event trigger. The first region sends a broadcast every 20 milliseconds to the client with sensor values. The second region receives the current sensor values from the server every 10 milliseconds with the method call [68].

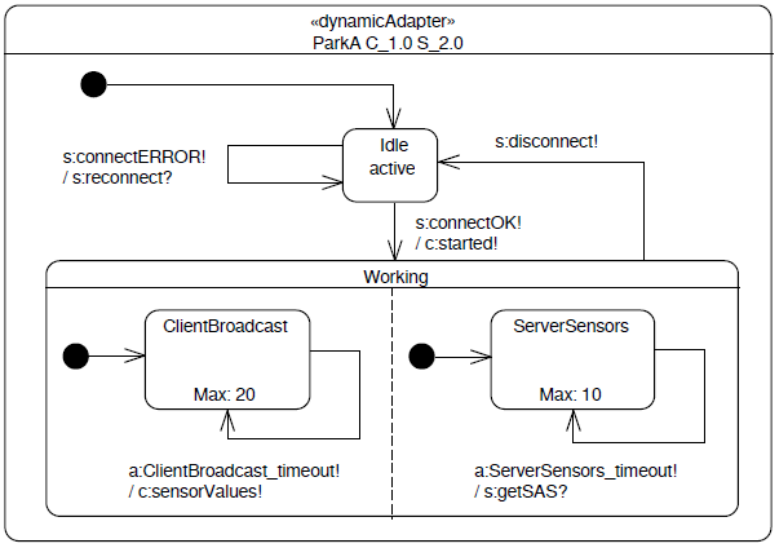


Fig. 22. State machine model for ParkA_1.0 and ParkA_2.0 event-driven communication pattern.

2.4.3 Android SWC Framework Event Chain Timing Analysis

Event dispatching in Android is orchestrated by the *Looper* class[77]. While *Looper* objects may be created by the developer, their common use is in the main thread of every Android application. As the application starts, the frameworks launch the main thread and runs a *Looper* that continuously dispatches *events*. *Events* that are dispatched by the *Looper* are called messages and each message defines a routine to be executed. Once a message routine is started, no other message routines can start until the first one completes. Additionally, the *Looper* class allows for a message to be enqueued both by the application and by the framework and supports removal of messages that may become obsolete before they are dispatched[77].

Timing analysis of *Looper* class can also be explained using state machine diagrams. The timing and order of message dispatching is controlled by the message type and parameters as follows:

- ***Delayed***(delay) denotes a message to be dispatched after a specified time elapse. If the time is set to 0 it denotes that the message should be dispatched directly after the current *event* finishes (assuming there are no other messages already scheduled before).
- ***At time***(when) denotes a message to be dispatched at a specific time in the *event* chain.
- ***Front*** denotes a message to be dispatched next. For messages of type *Front* even if the message queue already contains other messages scheduled for execution, the message will be scheduled before them.
- ***Idle*** denotes a message to be dispatched when the *Looper* has no other messages to dispatch.

2.4.4 ROS2 SWC Framework Event Chain Timing Analysis

From a logical perspective, ROS applications are composed of nodes, the smallest self-contained units of behavior. Nodes in ROS2 framework are mostly analogous in functionality to SWCs used in AUTOSAR, Franca and Android frameworks. In general, the nodes communicate using the *publish-subscribe* interface communication design pattern: nodes publish messages on topics, which broadcast the message to all nodes that are subscribed to the topic. Nodes react to incoming messages by activating callbacks to process each message.

For the analysis of the ROS application framework timing behavior, the focus is exclusively towards the simpler and more predictable single-threaded executor. The executor distinguishes four categories of callbacks: *timers*, which are triggered by system-level timers, *subscribers*, which are triggered by new messages on a subscribed topic, *services*, which are triggered by service requests, and *clients*, which are triggered by responses to service requests. The executor is responsible for taking messages from the input queues of the DDS (Data Distributed Service) layer (ROS middleware interacting with the client library) and executing the corresponding callback. All the event chains comprised solely of ROS callbacks are initially triggered by a timer [78].

```
#include <ros/callback_queue.h>
...
ros::CallbackQueue my_queue;
```

```
1 my_callback_queue.callAvailable(ros::WallDuration());
2 // alternatively, .callOne(ros::WallDuration()) to only call a single callback instead of
all available
3
```

```
1 ros::MultiThreadedSpinner spinner(4); // Use 4 threads
2 spinner.spin(); // spin() will not return until the node has been shutdown
3
```

Fig. 23. Overview of Callback queues and spins used for event chain for ROS2 framework .

Callback queues/spinning will influence the subscription queue, since how fast the *callbacks* are processed and how quickly messages are arriving determines whether messages will be dropped. The *CallbackQueue* class has two ways of invoking the callbacks inside it: *callAvailable*() and *callOne*() [78]. As illustrated in the Fig. 23[78][, *callAvailable*() will take everything currently in the *event* queue and invoke all of them and *callOne*() will simply invoke the oldest callback on the *event* queue. Both *callAvailable()* and *callOne()* can take in an optional timeout, which is the amount of time they will wait for a *callback* to become available before returning. If this optional timeout value is set to zero, then there are no callbacks in the *event* queue and the method will return immediately.

Chapter 3 Related Works

The Internet of Things, formerly known as IoT in vehicle domain is composed of a large and heterogeneous set of devices, software systems, and networks in the quest of intelligent environments. During the last decades, a myriad of heterogeneous protocols and standards have been used to address the requirements of this new technological approach. The number of protocols and standards are growing as the number of new applications increase. Consequently, one of the major challenges today in the Industrial IoT (IIoT) is the interoperability between systems with heterogeneous characteristics [6]. Great efforts have been made in recent decades to improve interoperability at semantic and syntactic levels. Streams of data were successfully transmitted between various vehicle services, however there was no meaning associated with the data [5].

To facilitate a holistic and meaningful data exchange between several heterogeneous service-based application components' interfaces in the vehicle domain, it is essential to link the data at semantic level using a shared vocabulary of the domain independent of platform specifications [8]. Most semantic alignment approaches in vehicle domain address this problem by semantically mapping the vocabularies of platform-specific interface concepts based on synergies in concepts at interface metamodel representation level.

3.1 Linking of Heterogeneous and Distributed Data at Semantic Level

In the automotive domain, there is a growing consensus that automotive domain services alone will not be enough to develop valuable and complex vehicle services or solutions due to the degree of heterogeneity, autonomy, and distribution of the services. Due to interaction of multiple heterogeneous knowledge domain enterprise applications to render services, several researchers agree that it is essential for automotive services to be machine understandable in order to allow the full deployment of efficient solutions supporting all the phases of the lifecycle of the services. It is always a good idea to is to encapsulate an application framework or an organization's functionalities within an appropriate standard interface and advertise them as automotive services [82]. Additionally, it is equally important that these interfaces are machine understandable and have proper meaning or semantics associated with them. The existing dominant standard for linking data of interfaces of automotive applications developed by different vendors are XML schemas. However, XML is for syntax with no semantics. Due to the widespread importance of integration and interoperability for intra- and inter-business processes, the research community in semantic web domain has tackled this problem and developed semantic standards such as the Resource Description Framework (RDF) and the OWL. RDF is a standard for creating descriptions of information. Therefore, RDF is a standard for semantics. OWL provides a language for defining structured Web-based ontologies which allows a richer integration and interoperability of data among communities and domains[82].

Authors et.al [63] propose XS2OWL transformation model that allows to transform the XML Schema constructs in OWL, so that applications using XML Schema based standards will be able to interoperate, exchange data by semantically linking them. The proposed XS2OWL transformation uses *XSLT Stylesheet* tools for transformation of the constructs from XML schemas to constructs of OWL-DL (OWL-Definition Language). Incorporating the XML and RDF paradigms approach was also proposed by Yin-Yang Web [60], however, the authors did not consider any heterogeneous sources with different syntax or data models in their proposed approach. They developed an integrated model for XML and RDF by integrating the semantics and inferencing rules of RDF into XML, so that XML querying can benefit from their RDF reasoner.

Although several tools have been implemented to generate semantically *Linked Data* from raw data, users still need to be aware of the underlying technologies and *Linked Data* principles to use them. Mapping languages enable to detach the mapping definitions from the implementation that executes them [67]. The *rmleditor* suggested by authors et. al [67], a visual graph-based user interface, which allows users to easily define the mappings that deliver the *rdf* (Resource Description Framework) representation of the corresponding raw data. Neither knowledge of the underlying mapping language nor the used technologies are required. The *rmleditor* aims to facilitate the editing of mappings, and thereby lowers the barriers to create semantically *Linked Data*.

3.2 Metamodel-based Modeling of SWC's Interface Models

In general, a metamodel defines the structure and semantics of the metadata. In conceptual modeling like metamodeling, most of the semantics is encoded with the attributes of classes and relation classes. It is thus interesting to analyze, for example how many and which kind of attributes have been introduced as an indicator for the complexity of the domain to be addressed by the modeling methodology [51]. The semantics of modeling languages is often not defined explicitly but hidden in modelling tools. To fix a specific formal semantics for languages, it should be defined precisely either in the metamodel specification or by transformations which transform software models into logic representations.

Metamodel-based modeling approaches are based on a staged architecture of models and metamodels, where the structure of lower-level models is defined by higher level metamodels. This staged architecture defines a layered structure, which is applied to define any domain-specific languages and general-purpose languages, e.g., UML, ecore, etc.

Also, for ontology technologies, the definition of metamodel enables the specification of model transformations of software engineering artifacts into OWL-related languages. For example, the transformation of ecore class diagrams into OWL uses transformation rules based on the metamodel of both languages. In context of enabling linked data capabilities to MOF (Meta Object facility) models, authors et. al [2] propose a TwoUse Toolkit. Software engineers are expected to use the TwoUse toolkit to extend UML or Ecore models with OWL annotations, to transform either of these metamodels into OWL and subsequently to query them, as seen in form of snippets of a Business Process Model Notation (BPMN) model in Fig. 24.

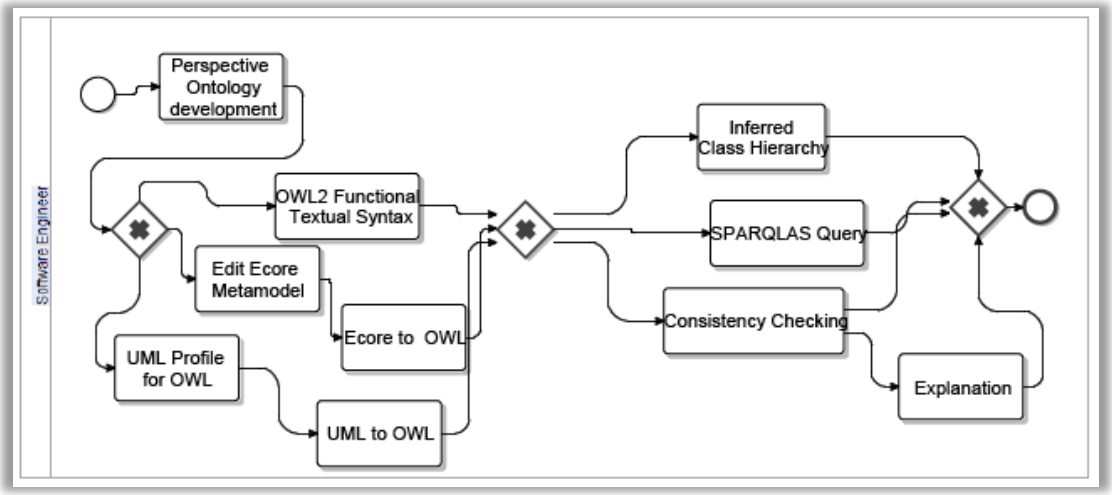


Fig. 24. Overview of BPMN model from TwoUse Toolkit using OWLizer for metamodel translations.

3.3 Semantic Mapping of Concepts of Interface Metamodels of SWC Frameworks for Interoperability

Semantics is the study of the meaning of signs, such as terms or words. Depending on the approaches, models, or methods used to add semantics to terms, different degrees of semantics can be achieved. In general, there are four major representations that can be used to model and organize concepts to semantically describe terms, namely: *controlled vocabularies*, *taxonomies*, *thesaurus*, and *ontologies*[82], as illustrated in Fig. 25.

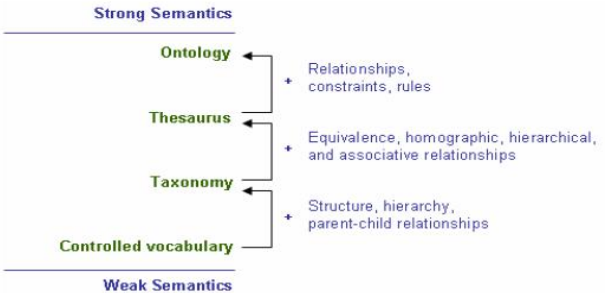


Fig. 25. Different levels of Semantics.

The aspects of interoperability as a general concept or approach covers technical, syntactic, semantic, and organizational issues, usually referenced as interoperability layers [83]. These layers, which are related and mutually interconnected, deal with the following objects:

- *Technical interoperability level*: signals, low-level services and data transfer protocols.
- *Syntactic interoperability level*: data in standardized exchange formats, mostly based on XML or similar formalisms.

- *Semantic interoperability level*: information in various shared knowledge representation structures such as taxonomies, ontologies, or topic maps.

- *Organizational interoperability level*: processes, defined as workflow sequences of tasks, integrated in a service-oriented environment.

The focus of this research report is on semantic interoperability; however, syntactic interoperability level is addressed as well. One of the most promising approaches to interoperability is the employment of semantic technologies [18], [1]. Semantics provides a capability to model and represent knowledge within a domain by means of explicit formalization of key domain concepts, their attributes and relations, as well as workflow sequences and structures. Considering the heterogeneous and distributed nature of the automotive domain, semantics can be very effectively used as a common background platform for describing the processes and services provided by automotive enterprises on various levels. The common platform then allows for integrating the services, making them interoperable and transparent for the end users, citizens and businesses. Additionally, a common representation language or vocabulary to describe the exchange of services using API models will allow to describe the services within the vehicle domain in a more standardized way.

3.4 Unified API Description/Specification Language for Vehicle Domain Application SWC Frameworks

Applications in vehicle domain are implemented as multiple distributed components, and those components call each other's APIs for the complete application to function. When developers design APIs for such applications, the solution characteristics they will typically prioritize are ease of programming for both the client and the server and efficiency of execution. Moreover, description of services in a language neutral manner is also vital for the widespread use of vehicle domain services. Service providers must describe their services and advertise them in a standardized way using a universal registry like commonly used in semantic Web domain, the Universal Description Discovery and Integration (UDDI), an XML-based registry for business internet services. The ability to access a correct description of service APIs is a known substantial need of both developers and automated tools [84].

WSDL is a unified web service description language used to describe concepts, related to Semantic Web services (SWS). Based on the analysis of WSDL descriptions, three types of web service elements can have their semantics increased by annotating them with ontological concepts using tools like WSDL-S are: operations, messages, preconditions and effects. All the elements are explicitly declared in a WSDL description [82]. From a modeling perspective, the Web Services Modeling Ontology (WSMO), provides ontological unified specifications for the description of semantic Web services. One of the main objectives of WSMO is to give a solution to application integration problems for Web services by providing a conceptual framework and a formal language for semantically describing all relevant aspects of Web services [83][82]. WSML (Web Service Modeling Language) is the language used to describe Web Service Modeling Ontology (WSMO) concepts, related to Semantic Web services (SWS). Authors et. al [87] proposes adapter-based approach to convert raw data like XML to WSML so as to use WSMO for modeling SWS for services interoperability, as seen in Fig. 26. This transformation method performs the conversion from the XML file into the WSML file with the help of a WSML template file.

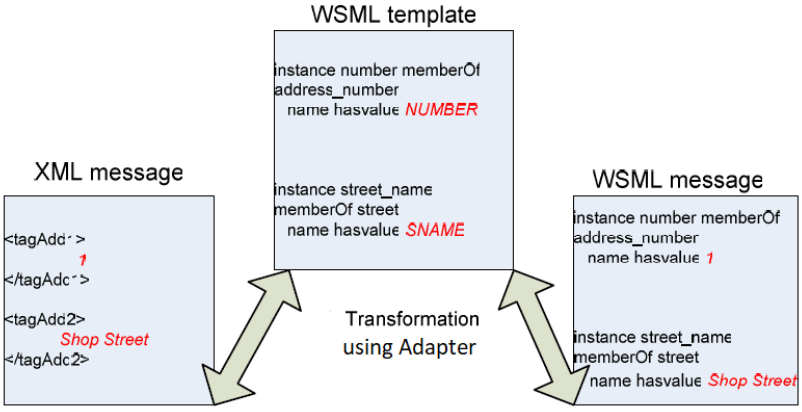


Fig. 26. XML to WSML transformation using Adapter.

In today's era, complex software systems are being built using the SOA and microservices paradigms. This also holds true for automotive software systems. The services, accessed by client applications through their APIs, mostly follow the remote operation call (RPC) oriented style or the representational state transfer (REST) style [84]. In the field of RPC-based web services, the WSDL of the SOAP technology provides a dominating, officially standardized, machine readable API specification with an underlying model. In theory, clean REST APIs would avoid the need for "APIs compatibility checks" by using the full Hypermedia as the Engine of Application State (HATEOAS) principle, enabling the client to explore the (evolving) API. This is one of the major reasons for the growing acceptance of OpenAPI specifications for API description in business domains like automotive. However, in practice, such checks are still needed either due to clients not traversing the hyperlinks or the APIs not following the principle sufficiently well [84]. In context of web service APIs compatibility, representation of service APIs in a unified manner independent of technology, authors et. al [84] adapted a model for service API representation, to facilitate subsequent processing in a unified manner.

3.5 MDE Vs Ontology Approach for Domain-Specific Interface Metamodels Semantic Alignments: Alternative or Complementary

Among artifacts produced by multiple modeling languages, MDE faces the following challenges towards semantic interoperability [5]: support for developers; interoperability among multiple artifacts and formal semantics of modeling languages. Addressing these challenges is crucial for the success of MDE. The authors of [27] present an automated approach to model-to-model mapping and transformation methodology, which applies semantic and syntactic checking measurements to detect the meanings and relations between different models automatically. The authors of [28] propose component model-to-model transformations to establish translation of semantics by manual mapping of programming languages of heterogeneous platforms [28].

In 2016, EAST-ADL was evolved as a DSM (Domain Specific Modelling) tool for automotive electronic systems. EAST-ADL is an ADL[37][22] (Architecture Description Language) aligned with AUTOSAR. EAST-ADL and provides a comprehensive approach for defining automotive electronic systems through an information model that captures engineering information in a standardized form [13]. Franca framework provides special support to those automotive domain IDLs which can be implemented using EMF (Eclipse Modelling framework) [6]. Based on the Franca core model, service interface specifications defined in other vehicle IDLs can be transformed to or from Franca framework. This feature of Franca framework could make it more suitable to support a generic automotive domain Meta IDL for future vehicle application software. AUTOSAR (Automotive Open System Architecture) standard for automotive has two platforms namely AUTOSAR Classic and AUTOSAR Adaptive. Both the platforms use a common exchange format or an IDL that is ARXML [8].

The intention to use a common IDL is to increase reuse of application software components and to ease cross-communication between the two automotive platforms. The authors of [11] advocate that language engineering technologies can be used for the interface definition and code generation to glue artifacts and to support the work within automotive heterogeneous development environments. The authors of [3] propose model-to-model transformations to be employed to establish translation of semantics of a DSL (Domain Specific language) by mapping it to those of low-level programming languages and platforms. There are already existing proposals to establish a bridge between message component interface models from ROS (Robot Operating System) and Google Protobuf (Protocol buffers) to overcome their cons and to obtain merged support from both ROS and Protobuf IDLs [5]. Such kind of bridges uses static semantic analysis of framework specific interface models as a knowledge base.

In contrast, issues like interoperability and formal semantics motivate the development of ontology web languages. Indeed, the World Wide Web Consortium (W3C) standard Web Ontology Language (OWL) [11][2], together with automated reasoning services, provides a powerful solution for formally describing domain concepts in an extensible way, thus allowing for precise specification of the semantics of concepts as well as for interoperability between ontology specifications. Ontologies consist of definitional aspects such as high-level schemas and assertional aspects such as entities, attributes, interrelationships between entities, domain vocabulary and factual knowledge, all connected in a semantic manner. Ontologies provide a common understanding of a domain. They allow the domain to be communicated between people, organizations, and application systems. Ontologies provide the specific tools to organize and provide a useful description of heterogeneous content [82].

In the perspective of service interoperability, in the semantic Web domain, OWL-S is an ontology within the OWL-based framework, which is used in conjunction with domain ontologies specified in OWL, provides standard means of specifying declaratively APIs for Web services that enable automated Web service interoperability [36]. The Web Service Modeling Ontology (WSMO) is a conceptual model that provides an ontology-based framework

which supports deployment and interoperability of services for semantic Web domain. However, applying WSMO perspective in automotive domain environment for services interoperability remains challenging and unanswered [35].

UML class-based modeling and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, types, generalization, and instances [1][2]. Despite of the similarities, both approaches present restrictions that may be overcome by an integration. Since both MDE modelling based approaches and ontology-based approaches provide complementary benefits, contemporary software development should make use of both. The question remains is *how model driven software development and ontology driven software development can be amalgamated so that the model driven developers can use a familiar environment and advantages of the ontology driven development are also additionally available*? [2]. The possible benefits by amalgamation of MDE and ontology paradigms are: Firstly, it provides software developers with more modeling power. Secondly, it enables semantic software developers to use object-oriented concepts like inheritance, operation and polymorphism together with ontologies in a platform independent way.

Authors of [2] specifies a coherent framework for integrated usage of both concepts and formal semantics of modelling language, comprising the benefits of both UML-based modelling and OWL [3]. Although the authors provide the theoretical foundations of such transformation, but the practical use of such transformations cannot be implemented with current versions of Eclipse Modelling Frameworks (EMFs). The authors of [12] proposes alignment of ontologies of source UML models with semantic heterogeneity into a single ontology or merged coherent model by using a process of detection and resolution of semantic conflicts that may exist among the different UML models.

The OWLizer project [26] in 2016, attempts to translate Ecore metamodels to OWL ontologies, the handcrafted approach to transform Ecore metamodels to ontologies, using ABox (Instance level) and TBox (Schema Level) reasoning [3], is however, overly complex and is not supported any more with the current versions of EMFs. The EMF4SW eclipse plug-ins provide OWL to Ecore and Ecore to OWL transformations capabilities [7] [9]. However, no contribution has been made to the project since 2012. The authors of [24] propose to invent a UML profile, which allows to visually model OWL ontologies in a notation that is close to the UML notation. This contribution is not supported with proper case studies for practical implementation. Rahmani et al. [25] describe an adjustable transformation from OWL to Ecore and identified that it is possible to represent most OWL constructs with Ecore and OCL invariants. However, such a transformation has the purpose of aligning OWL constraints with Object Constraint language (OCL) invariants and does not cover OWL reasoning services like realization and instance checking.

Both MDE and ontology technology paradigms uses conceptual metamodels to identify semantically related modeled entities or concepts. Understanding the role of ontology technologies in MDE like domain conceptualization using shared vocabulary, automated reasoning, inferred axioms, dynamic classification and consistency checking is essential for leveraging the development of promising and complete solutions to semantic interoperability for practical implementation. When trying to extend MDE based metamodels to ontological metamodels, the gap between the implementation-oriented focus of MDE based metamodels and the knowledge representation focus of ontological metamodels must be closed.

3.6 Evaluation of Interface Metamodels Semantic Alignment Quality for Vehicle Application SWC Frameworks

The measure of success of both MDE and ontology paradigms' metamodeling approaches or methodologies towards tackling semantic interoperability issue could be however estimated by evaluation of metrics such as qualitative, quantitative, etc. Related work in the context of quality metrics for meta models can be categorized into two groups: Firstly, work that deals with quality on the meta level, i.e., on meta models. Secondly, approaches that address quality on the instance level, i.e., on models [48]. Metamodels which plays a pivotal role in conceptual modeling as they manifest the abstraction level applied when creating conceptual models. Consequently, design decisions made by the metamodel developer determines utility, capabilities, and expressiveness of the conceptual modeling languages (for example ecore, OWL2), modeling methodologies and eventually the created models. However, very few researches define and applies proper metrics for analyzing the structure and capabilities of metamodels, and eventually support the development of new metamodels. This not only concerns general-purpose modeling languages, but also domain-specific ones, which usually undergo shorter update cycles [51].

Based on literature artifacts, it can be argued that most existing ontology metrics in automotive domain are based on structural notions without considering the semantics which leads to incomparable measurement results in

practical implementations. In principal, evaluation of depth of semantic alignment for the interface ontologies using quality metrics is substantial to guarantee that it meets the vehicle application communication requirements for semantic interoperability.

Despite the great effort, the alignment and merging systems ontologies are still semi-automatic, which reduces the burden of creating and maintaining manual applications. These systems require human intervention to validate the alignment and merge ontologies. In addition, they use various aids, such as the common vocabulary, reference ontology, etc. [25] to detect mapping candidates. The validation process after the detection of initial mappings helps to find inconsistent mappings; which is usually done by domain experts and performed manually in most cases. During the validation phase, the domain expert is responsible for classifying mappings results from alignment based on a similarity measure in equivalence mapping and is-a mapping [UMLintegration].

Aligning semantic ontologies based on semantic synergies to support interoperability represents a great interest in automotive application domain that manipulate heterogeneous overlapping knowledge frameworks. The alignment produces a set of mappings that can be used in the merging phase, but before that, a process of validating must be conducted in order to find incoherent and inconsistent mappings and keep only valid ones. This validation step is highly required to produce the correctness and consistency of final integrated model.

3.7 MDE and Ontology Modeling Approaches to tackle Semantic Interoperability between Vehicle Component Framework Interfaces

The applications of todays' cars can be generally clustered into different sub-domains like infotainment, advanced Adaptive Cruise Control, connectivity, Dynamic Short-Range Communication (DSRC), etc. Collaboration between apps of these heterogeneous vehicle architectural domains is highly demanded to accomplish the novel automotive services, such as autonomous driving, V2V (Vehicle-to-Vehicle communication), etc. Nevertheless, semantic interoperability between the vehicle apps stands as a major challenge. MDE (Model Driven Engineering) and ontology technology can be viewed as two software paradigms for addressing semantic alignment solutions to tackle semantic interoperability. A traditional model-driven approach mostly aims at finding the semantic mapping directly starting from the two models, say Platform Independent Model (PIM) A and PIM B, deriving then the PIM2PIM mapping. Whereas, the ontology-based approach does it indirectly, by means of a reference Ontology. Therefore, in the later case, the semantic mapping is obtained by the composition of two partial PIM2ONT and ONT2PIM mappings, as illustrated in Fig. 24.

3.7.1.1 Model Driven Engineering (MDE): The Rationale

The Model-Driven Architecture (MDA) has been proposed by the Object Management Group (OMG), which describes their vision of MDD (Model Driven Development). They describe Model-Driven Engineering (MDE) as a superset of MDD, because it goes beyond development activities and includes other modeling tasks in the complete software engineering process [3]. In MDE, models are described by modelling languages, where modelling languages themselves are described by so called metamodeling languages [2]. Besides models and metamodels, MDA describes another key element of MDE that is model transformations, which define mappings between different source and target models.

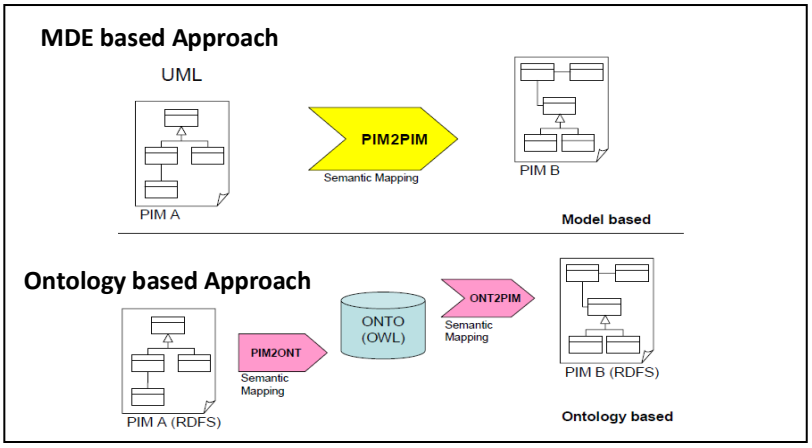


Fig. 27. Comparison of MDE and ontology based approaches for platform-specific model to model semantic mapping.

Each source and target models could further confirm to source and target metamodels respectively. Model transformations work by specifying transformation rules. An example of source metamodel to target metamodel transformation is illustrated in Fig. 27 [3].

In general, MDE consists of the following two major artifacts: firstly, modelling languages, which are used to describe a set of models and lastly, model transformations, which are used to translate models represented in one language into models represented in another language of at least one concrete syntax and semantics.

Software technology is constantly changing, for instance, due to new middleware platforms. As a result of the intertwining of the computing and the problem solution, software engineers often spend considerable amounts of time to port applications to different target platforms, or newer versions of platforms. Moreover, if only the computing solution exists, knowledge about the problem solution might get lost when the developer of a system leaves the organization [2]. To tackle this, MDE aims to solve software integration, interoperability and maintenance problems, by separating functionality, or behavior, from specific platforms and technologies. MDE separates platform independent models (PIM) and platform-specific models (PSM) [18]. PIMs are formal specifications of the structure and behavior of a system. MDE claims that by using PIMs that abstract away platform-specific details, integration and interoperability across various platforms should become easier to produce.

The shift from code-centric to transformation-centric software development places models as first-class entities in model-driven development processes [5]. However, due to a lack of interoperability, it is often difficult for MDE tools to combine the artifacts produced by heterogeneous modelling languages when integrating different software models, thus the potential of model driven software development cannot be fully exploited. Moreover, a direct integration of different modeling languages by their metamodels is not a trivial task, and often leads to handcrafted solutions created in an error-prone process usually inducing high maintenance overheads [2]. Metamodels prove to be more implementation oriented as they often bear design decisions that allow producing sound, object-oriented implementations. Due to this, language concepts can be hidden in a metamodel [5] [4].

3.7.1.2 Ontology Technology: The Rationale

The meaning of a model must be well-defined such that multiple developers can understand and work with it. If the meaning or semantic is not clear there would be no possibility to define automated transformations from one abstract model to a more specific model. Formal semantics and constraints are the basis for interoperability and formal domain analysis. Modeling issues like formal semantics or interoperability motivated the development of ontology languages [4]. Formal semantics constrain the meaning of models, such that their interpretations are the same for different persons (or machines).

Ontologies provide shared domain conceptualizations representing knowledge by a vocabulary and, typically, logical [9]. Among ontology languages, build on the W3C standard, Web Ontology Language (OWL) stands for a family of languages with increasing expressiveness. OWL2, the emerging new version of OWL is a highly expressive language with practically efficient definitions. OWL2 features many types of axioms and thus provides different constructs to restrict classes or properties. OWL provides class definition language for ontologies and implied logical constraints on the properties of their members. OWL allows for deriving concept hierarchies from logically defined class axioms stating necessary and enough conditions of class membership [2][1]. The logics of class definitions may be validated by using corresponding automated reasoning technology.

In general, ontologies are used to define sets of concepts that describe domain knowledge and allow for specifying classes by rich, precise logical definitions. The difference between OWL and modeling languages such as Unified Modelling Language (UML) class diagrams is the capability to describe classes in many ways and to handle incomplete knowledge. These OWL features increase the expressiveness of the metamodeling language, making OWL a suitable language to formally define classes of modeling languages [4].

UML class-based modeling using MDE and OWL class-based modelling comprise many constituents that are similar in many respects like classes, associations, properties, packages, types, generalization and instances, however, despite of commonalities, both modeling languages presents restrictions such as UML class-based modelling allows only static specification of specialization and generalization of classes and relationships, whereas OWL provides mechanisms to define these as dynamic specifications [1][24].

Ontology engineering uses multiple languages for defining services. For example, modelers of ontology matching services need to manage different languages: (1) an ontology translation language to specify translation rules and (2) a programming language to specify built-ins, when the ontology translation language does not provide

constructs to completely specify a given translation rule. This intricate and disintegrated manner draws their attention away from the alignment task proper down into diverging technical details of the translation model. Addressing this issue allows developers to concentrate on constructs related to the problem domain, raising the abstraction level [9]. Moreover, by defining domain concepts as first-class citizens, developers may reuse these domain concepts on different target platforms. It helps to improve productivity, since modelers will not have to be aware of platform-specific details and will be able to exchange translation models even when they use different ontology platforms.

3.8 Comparison of Author's Contribution to the State of the Art

The research work presented in this contribution has been communicated to outside world through conference papers, a journal paper and conference workshops. The following list represents the published papers along with a brief description.

- *S. De, M. Niklas, J. Mottok and P. Brada, "Semantic Synergy Exploration in Interface Description Models of Heterogenous Vehicle Frameworks: Towards Automotive Meta Interface Description Model", ARCS Workshop 2019; 32nd IEEE International Conference on Architecture of Computing Systems, Copenhagen, Denmark, 2019, pp. 1-8.*

In the direction of semantic interoperability, vehicle component frameworks can be compared and correlated across different vehicle programs, thus establishing the foundation for overall value creation efficiency, interoperability, increased reusability and ultimately increased innovation. This can be done by comparing the semantics traits of IDLs of various SOA based vehicle application frameworks, using semantic mappings and to explore possible synergies or commonalities among their semantics traits. The interface semantic synergy traits could be further abstracted and generalized to define the generic domain SWCs' interface traits that could be considered by Domain Specific Language (DSL) designers for evolution unified domain solutions like automotive domain Meta IDL in the future [13].

- *S. De, M. Niklas, J. Mottok, and P. Brada," A Semantic Analysis of Interface Description Models of Heterogeneous Vehicle Application Frameworks: An Approach Towards Synergy Exploration". DOI: 10.5220/0007472503940401, In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pages 394-401, ISBN: 978-989-758-358-2.*

Despite of syntactic differences between platform-specific IDLs' syntax, the approach to correlate these IDLs based on semantic synergies among their traits could provide the knowledge base for the increase in the interoperability at application interface code generation level, increase in overall efficiency and development of an automotive domain generic interface software solutions, by facilitating coexistence of components of heterogeneous frameworks in the same ECU HPC software platforms future vehicle software [14].

- *S. De, M. Niklas, J. Mottok and P. Brada, "Model Transformation of Application Software Component from Classic to Adaptive AUTOSAR: An Approach to Migrate Software Components", 44th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) Workshop, Prague, Czech Republic, August 29th - 31st, 2018.*

Based on semantic commonalities between API specifications of heterogeneous vehicle application frameworks' IDLs, the SWC models of different automotive platforms such as software platforms of AUTOSAR (Automotive Open System Architecture) standard, namely AUTOSAR Classic and AUTOSAR Adaptive can be model transformed from one platform to another in a bidirectional way at application SWC level [15].

- *S. De, M. Niklas, B. Rooney, J. Mottok and P. Brada, "Towards Translation of Semantics of Automotive Interface Description Models from Franca to AUTOSAR Frameworks: An Approach using Semantic Synergies", 2019 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 2019, pp. 1-6, doi: 10.23919/AE.2019.8867018.*

The software architecture of future vehicle sub-systems' complex and high-performance ECUs intends to evolve multifarious high-level domains such as Telematics, Infotainment, Robotics, etc. to support complex vehicle services. Such domains are not covered by the de-facto standard of automotive system architecture, AUTOSAR. Therefore, the semantic gap between component frameworks of AUTOSAR and such other standards in vehicle application domain could be bridged using semantic analysis and mapping between their IDLs to improvise the service interoperability between them, when co-existing in the same vehicle ECU. The above-mentioned technical

paper proposes to bridge the semantic gap between AUTOSAR Adaptive framework interface metamodel and infotainment Franca framework interface metamodel by using one-to-one semantic mapping of the framework specific IDL constructs, based on semantic overlapping of interface functional concepts [17].

- *S. De, M. Niklas, B. Rooney, J. Mottok, P. Brada, "Towards Semantic model-to-model Mapping of Cross-Domain Component Interfaces for Interoperability of Vehicle Applications An Approach towards Synergy Exploration", In: CEUR Workshop proceedings, ModComp, Vol. 2442, Munich Germany, 2019.*

Manual semantic checking measurements for semantic analysis at an application component communication interface level to understand the meanings and semantic relations between the different metamodel specifications of cross-domain component frameworks' interface models within vehicle domain based on explorations of semantic commonalities, to ensure that interface description models of software components from heterogeneous framework can be compared, correlated and re-used for automotive services based on semantic synergies. This was the major contribution of paper mentioned above [8].

- *S. De, M. Niklas, B. Rooney, J. Mottok, P. Brada, "Semantic Mapping of Component Framework Interface Ontologies for Interoperability of Vehicle Applications", Elsevier Procedia Computer Science. 170. 813-818. 10.1016/j.procs.2020.03.151, 2020.*

Considering the components' interface models as data models, in the automotive domain, XML schemas are the preferred standard for interface description exchange between various enterprise application component framework templates, e.g. AUTOSAR Classic, AUTOSAR Adaptive, Franca, ROS, etc. To confront the issue of interoperability by exploring the possibilities of semantic alignments between heterogeneous interface data models with heterogeneous semantic traits, the paper at [7] **Fehler! Verweisquelle konnte nicht gefunden werden.** proposes a solution to tackle semantic data heterogeneity by schematically translate the XML schemas representing the various component interface data models to RDFS. RDFS are ontology-based schemas that can be represented as an object model or a kind of constrained relational model.

The overview of what is covered by the related works mentioned in this subsection is summarized in TABLE II.

TABLE II. OVERVIEW OF COVERAGE OF RELATED WORKS DONE IN DIRECTION OF CURRENT RESEARCH

References of related work used in this section	Static Interface Semantic Analysis Type (including IDLs, model-to-model mapping for semantic translation)	Challenges of MDE based Metamodeling approach and Strengths of Ontology towards semantic interoperability	Integration of MDE and ontology-based Metamodeling methodologies	Solutions towards DSM, DSL or equivalent solutions for semantic interoperability	Metrics Evaluation to measure quality of MDE and/or Ontology metamodeling methodologies /approaches
[5]			✔		
[27]	✔				
[28]	✔				
[37]				✔	
[1]		✔	✔		
[2]		✔	✔		
[3]			✔		
[11]		✔			
[12]			✔		
[9]				✔	
[7]	✔				
[36]				✔	
[35]				✔	
[48]					✔
[51]					✔
[24]			✔		
[25]			✔		

[26]			✔		
Authors' Contribution to the State of the Art					
[14]	✔				
[15]	✔				
[8]	✔				
[13]				✔	
[18]	✔				
[17]	✔				

Part II Analysis Level

The analysis level represents study of the semantics of API architecture styles of heterogeneous vehicle component frameworks. This level includes semantic analysis on various interface semantic traits of SWCs of vehicle application domain frameworks, as illustrated in Fig.26. This level also includes information on semantics of API design styles or patterns employed by vehicle application frameworks for data exchange between components' interfaces for communication at application SWC model level.

The analysis level also includes semantic analysis of interface run-time behavior model, explicitly describing the *timing constraints* applicable to SWCs' interfaces represented by *Events* or *Event Chains*. The *Event Chains* are triggered based on time or data arrival on ports.

The semantic analysis of component frameworks' API architectures uses communication interface semantic traits analysis at M1 and M2 layers of OMG (Object management Group) standards' *four-layered modelling architecture.* That means, both at model and code generation level (associated with the IDLs), the APIs of SWCs (or equivalent representations) in vehicle application frameworks are semantically analyzed at model and metamodel levels.

At the modeling architecture layer M1, API architecture blueprints of real-world vehicle domain case study has been used for illustration purpose. Semantic analysis and comparisons of generated codes associated with vehicle frameworks' IDLs uses the M1 layered vehicle domain case study to explore semantic synergies in their interface concepts despite of the syntactic differences in the interface specification languages.

The intention to explore semantic synergies among vehicle application component frameworks' IDLs, interface description models and metamodels is to correlate the heterogeneous application frameworks' components and to ensure reuse of these SWCs through their interfaces for example, when semantic integrated to evolve future holistic software solutions.

The analysis level as seen in Fig. 28, attempts one step forward to tackle semantic interoperability by presenting a hypothesis on abstract functional layered architecture towards the evolution of a unified interface description software solution for vehicle application frameworks from a functional point of view based on analysis of existing artifacts.

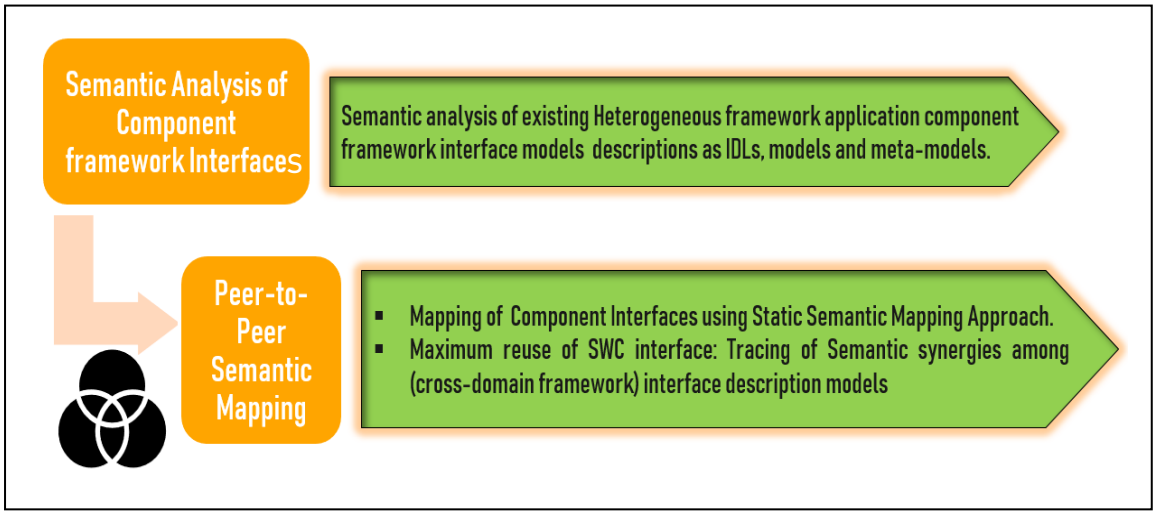


Fig. 28. Overview of Analysis level.

Chapter 4 Survey of Vehicle Domain Interface Description Languages (IDLs): Identification of Semantic Commonalities

As in many other engineering disciplines, automotive software engineering is well suited to collaboration. Having different perspectives and diverse experiences strengthens engineering projects. Automotive application software component modelling is a fundamental aspect of the automotive industry and is becoming increasingly collaborative. Since automotive application software component modelling become more and more complex, in order to increase productivity, modeling is necessary to bridge the gap between business and technology. In this sense, models allow the domain problems to be described by using the terms that are familiar to domain experts rather than terms that are only familiar to specific technical experts.

The automatic semantic integration of UML class diagrams derived from different automotive platform component framework interface models involves detecting the semantic, syntactic, and structural relationships. Semantic similarity measures play an important role for detecting different relationships in order to ensure alignment and merging of class diagrams. Most components interfaces alignment approaches address this problem by calculating the similarities between entities of interface models (such as concepts, roles, etc.) and produce candidate alignments based on the similarities obtained by comparing the entities one by one.

4.1 Semantic mapping of Component Framework IDLs: The Rationale

Over the past few years, several IDLs from domain specific and domain agnostic frameworks in vehicle application domain are being used for specifications of various SWCs for complex vehicle apps, e.g., autonomous driving, etc. Usually, it is observed that a given vehicle application SWC model has a commonality in fundamental semantics, despite using different syntactic representation, when modeling the same vehicle application interface in different frameworks using different IDLs. Streams of interface data are successfully transmitted between frameworks of cross-enterprise vehicle domain subsystems, however there is no meaning associated with the data. These further causes a decrease in efficiency and reusability of vehicle application SWCs. In vehicle domain there can be an overwhelming number of ways on how to implement even a simple two-way communication using legacy or open-source platform specific framework tools.

To ease the semantic data heterogeneity caused due to heterogeneous artifacts and different vocabularies for terminologies used by various platform specific IDLs in vehicle domain by application SWCs, it is essential to trace and identify semantic synergies between the SWCs' interface concepts. That is, to enable semantic interoperability between heterogeneous vehicle components' interfaces as seen in the Fig. 29. for an IoT usecase *Vehicle Position Finder*, it is time to focus on exploration of semantic synergies among the communication interface traits of the vehicle apps' heterogeneous SWCs frameworks IDLs from modeling perspective. This further helps in filling the semantic gaps between the component interface models for cross-enterprise collaboration in vehicle domain as seen in Fig. 29.

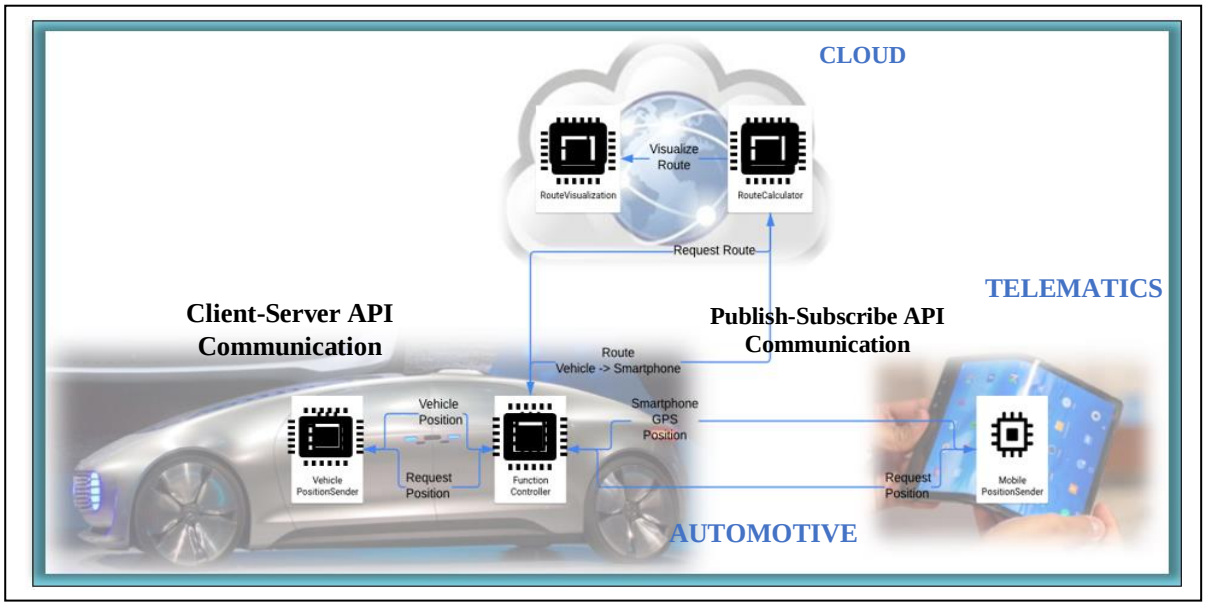


Fig. 29. Overview of heterogeneous modes of communication between outside world and vehicle for IoT software solutions.

The classification using semantic survey of framework IDLs does not show the success of a specific framework IDL or any specific business model within the automotive application domain, but it is based purely on the statical analysis of technical traits of the IDLs supported by various vehicle application platforms. Additionally, from interoperability viewpoint such a classification and comparison of IDLs can help future domain experts to understand better the areas of conceptual semantic synergies among the cross-enterprise application SWC frameworks interface traits and decide which all interface semantic traits of these SWCs frameworks can be generalized when semantically integrating to a vehicle domain specific global interface software solution in future.

In the current state of automotive Industry, in the case of an ECU with multiple CPU core partitions, there is a high probability that one platform-specific framework specific IDL model and one communication protocol used for deployment of APIs, will not be optimum for all the ECU core partitions and for all kinds of peer communication. Consequently, it has become necessary to combine the software components and subsystems as well as message formats from different platforms of the automotive domain, to provide cross-platform functionalities. For a vehicle application software component of a frameworks running in one partition of a high performance vehicle ECU to communicate with the an application component of a different framework in another partition of ECU, requires a generic domain-specific interface model representation template for interaction which would remain transparent in its representation format among all the communication peer partners and could be employed for the deployment of the API models to all the partitions independent of platform specific deployment description.

Proven from the past innovative contributions on DSM (Domain-specific modelling), the semantic alignment and integration of metamodels of component interface models can be made easier, when shifting the focus from interface modelling language-centric to concept-centric by raising the level of abstraction beyond platform-specific interface modelling language implementation and specifying the solution directly using domain concepts [1][3].

4.2 Semantic Comparison of Vehicle Domain Cross-enterprise Platforms Component Frameworks IDLs

The semantics of a modeling language allows for determining the truth value of elements in the model with respect to the system being defined. In other words, the semantics of a modeling language provides the meaning to its syntactical elements by mapping them to a meaningful representation. For example, the UML specification defines the semantics of the UML language by explaining each UML modeling concept using natural language. In other words, in model-driven engineering, metamodels serve as abstract syntax, whereas models serve as snapshots of languages.

Current System Engineering models in an automotive domain such as SysML (System Modelling Language), UML, etc. allows graphical modelling of component interfaces independent of software. Typically, an IDL defines the software interface agreements between the application component interfaces. IDLs are typically bound to one or more programming language generators. Over the time, in the automotive application domain the level of abstraction at which functionality is specified, published and or consumed has gradually become higher and higher [16]. Eventually progress has been made from modules, to objects, to components, and now to services [16]. A service is the major construct for publishing and should be used at the point of each significant interface.

Today most of the software component interfaces are based on Service Contracts, thereby allowing heterogeneous systems to communicate and interchange their services. The SOA (Service-Oriented Architecture) pattern allows us to manage the usage (delivery, acquisition, consumption, etc.) in terms of, related services [16]. To bridge the semantic gap between the vehicle application SWC frameworks and to achieve interoperability among them by reusing of artifacts and correlating the frameworks' SWCs requires better understanding of the semantic mapping at application SWC interface level [1][9].

The Approach to Semantic Mapping of Vehicle Domain IDLs

To semantically analyze the IDLs of different platform SWC frameworks' IDLs based on interface traits or fundamental characteristics, it is necessary to consider some platform-independent vehicle domain-specific generic SWC interface semantic traits which could be considered as the knowledge base for the frameworks' IDLs semantic comparison from a modeling perspective. These generic, abstract interface semantic functional traits basically include:

- *Interface type*: The distinction of the basic interface type: operation-based (e.g. methods invocations) and data-based service interface (e.g. data passing), etc.

- *Separation of Interface Roles:* The distinction between the provider and the consumers-part of a service interface.
- *Interface interaction points*: Service interface interaction points (e.g. ports, topics, etc.) at software components interaction level.
- *Data exchange Method Calls*: Method signatures containing information with valid parameter types, e.g. *ClientServer, Sender-Receiver, Publish-Subscribe, Broadcast* etc.
- *Method Calls Behavior*: Synchronous, Asynchronous, etc.
- *Attributes*: Specification of attributes or fields e.g. getters, setters, Notifiers, etc.
- *Data Types:* Different ranges of various datatypes e.g. int, float, string, array ,etc.
- *Interface Communication Design Pattern*: RPC(Remote Procedure Call), gRPC, REST, etc.
- *Optional Interface Binding and Middleware Protocol:* The binding type describes the way a vehicle application SWC interfaces binds to a middleware communication protocol for intra- or inter-ECU communication.

The vehicle domain-specific generic, abstract interface semantic traits in a way describes or represents the semantic traits of the concrete cross-enterprise platforms application frameworks' IDLs existing in the vehicle domain. These interface semantic traits provide no details bound to the specifics of functionalities of the concrete frameworks' SWC models and only provide a shallow understanding of the SWC from interface perspective. Exploration of interface semantic traits synergies between the frameworks' IDLs is useful for the holistic service information exchange.

In Fig. 12, M0, M1, M2 and M3 represent different model abstraction levels based on OMG standards (for more details refer Chapter 2). Fig. 31 illustrates a hypothesis for evolution of an abstract Meta IDL model for vehicle domain using *Multi-level modeling Approach*. To begin with the model abstraction levels, that is layer M0 includes a real-world typical vehicle domain case study is considered for the comparative analysis of interface traits at semantic level at layer M1. The *Generic API Repository* that is illustrated as layers below M0 abstracts the raw vehicle application interface models from the framework bound specific communication protocol syntax. As a result, the vehicle application would reach any services independent from its deployment. Layer M1 includes the semantic analysis of legacy and open-source vehicle platforms frameworks SWC interface models to identify the synergies in semantic communication interface traits between these heterogeneous platform specific SWCs interfaces' descriptions or IDLs both at model and code level. A typical vehicle domain case study from layer M0 can be considered for such semantic analysis. Layer M2 defines a *Generic Traits Repository* which includes all the platform-independent, vehicle domain-specific generic SWC communication interface semantic traits described above in this subsection. Layer M2 forms the knowledge base for the semantic analysis of heterogeneous platform specific SWCs frameworks' communication interfaces at layer M1. From a future work perspective, a hypothetical language model of Meta-Interface Description Language (Meta-IDL) is defined both at code and model level at layer M3, based on the abstraction of vehicle domain-specific *Generic Traits Repository* of SWCs' communication interface traits at layer M2.

The interface semantic analysis approach at layer M1 is based on *functional* and *non-functional* communication interface traits of vehicle domain application component frameworks. The approach is illustrated using a simple vehicle domain case study, named *SeatHeating* SWC. For demonstration of the approach, a realization of the abstract communication interface model of the considered case study is considered for each of the vehicle domain platform specific SWC frameworks by using concrete IDL alternatives (in detail subsection 5.2.2). The *SeatHeating* software component is a sensor actuator component model used in vehicle power management functional cluster to monitor seat heat. The symbolic figure for *SeatHeating* software component is depicted by Fig. 30.

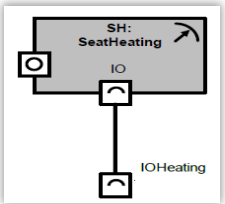


Fig. 30. Symbolic representation of SeatHeating SWC.

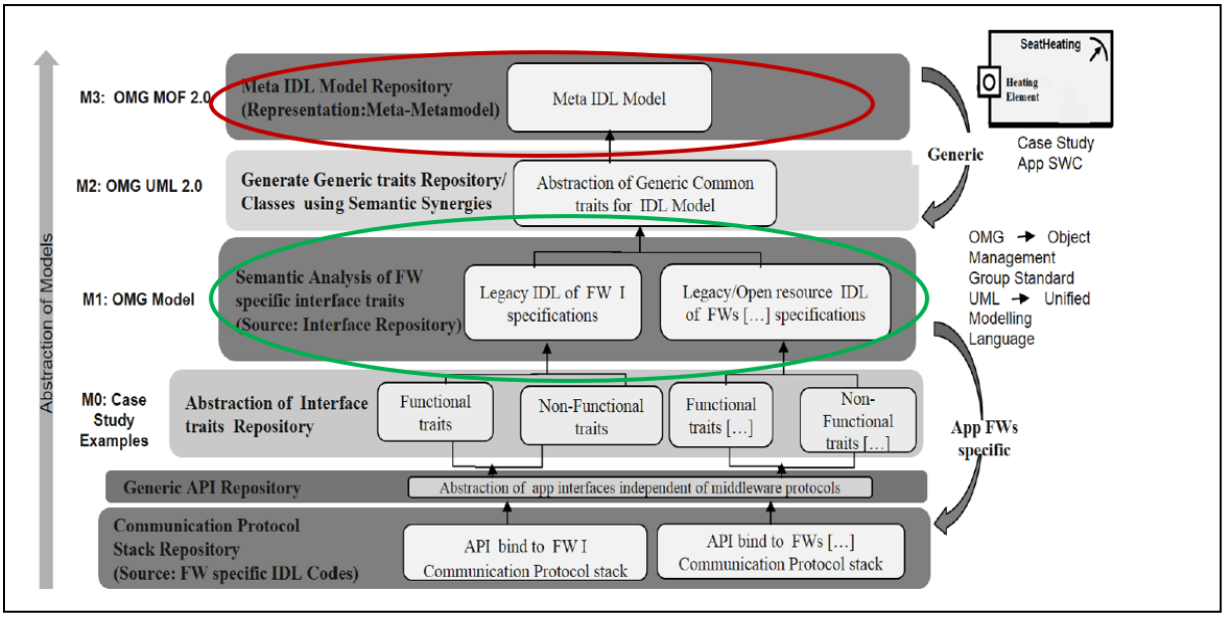


Fig. 31. Multi-level modeling approach for evolution of abstract Meta IDL model for vehicle domain.

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 License supported.
- Language Bindings supported.

4.2.1 Demonstration of the Approach using Vehicle Domain IDL Alternatives

In consideration to Fig. 31, this section provides an overview of the existing automotive IDL alternatives of SWC frameworks and additionally includes discussion on middleware communication protocols used by these automotive domain frameworks for deployment of the APIs. The heterogeneous vehicle platforms and knowledge domains component framework specific abstract IDL models discussed in this section are based on *SeatHeating* software component model case study, as described in the earlier subsection.

4.2.1.1 Automotive Domain: AUTOSAR Adaptive Framework IDL: ARXML

The ARXML (AUTOSAR eXtensible Markup Language) or AUTOSAR XML is the standard description format used to model all AUTOSAR software component models related to the AUTOSAR Classic platform and the AUTOSAR Adaptive platform. The AUTOSAR application SWC template metamodel is represented using ARXML and is validated using an XML Schema [41].

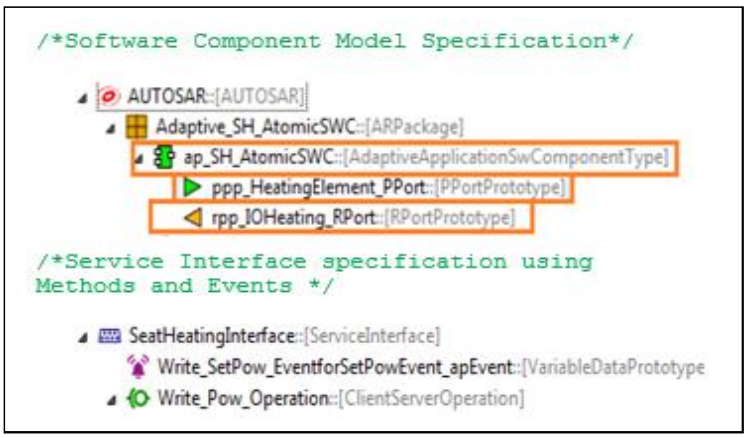


Fig. 32. Illustration of the case study SWC communication interface model using ARXML.

One of the major benefits of using ARXML as IDL is to simplify the comparison of AUTOSAR SWC descriptions from different AUTOSAR based automotive platforms. This enables interoperability among different AUTOSAR platforms such as AUTOSAR Adaptive and AUTOSAR Classic. Fig. 30 illustrates a *SeatHeating* SWC model using ARXML. An AUTOSAR Adaptive SWC provider port (*PPortPrototype*) and receiver port (*RPortPrototype*) interfaces are illustrated in the Fig. 32[13].

The service based SWCs interface model of AUTOSAR Adaptive automotive platform uses RPC communication design pattern. In specific, the services based SWC interfaces of AUTOSAR Adaptive platform uses SOME/IP (Scalable service-Oriented Middleware Over IP) protocol stack as a middleware solution.

Infotainment Domain: Franca (and Franca+) Framework IDL: FIDL

Franca IDL (FIDL) is developed as a part of the GENIVI standard Franca (version 0.13.0) framework and supports IVI (In-Vehicle infotainment) systems' interfaces. Franca IDL (FIDL) is language binding neutral IDL and independent of concrete bindings. APIs defined with Franca IDL consist of collections of attributes, methods and broadcasts [16][13]. Franca+ IDL (FCDL) provides an extension to the native Franca framework IDL that adds support to the modeling of components, composition of components, typed ports (provides and required), Port interfaces (optional major and minor versions) and connectors between ports. Franca framework uses FIDL to define application interfaces and FCDL to define application SWCs and their configurations.

Using Franca IDL, a vehicle application client calls the backend server using a vehicle ID and a struct (Structure) defining service details such as a unique Service ID. Fig. 33 illustrates the case study on *SeatHeating* vehicle application using Franca IDL. In the context of versioning, Franca IDL has backward compatibility. The APIs of Franca framework is usually compatible with RPC (Remote Procedure Call) communication design patterns using various middleware protocols such as SOME/IP (Scalable service Oriented Middleware over IP) .

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

/*Generated Interface for Client(C++)*/
class SeatHeatingServProxy {public void
subscribeSeatHeatingServ (const uint32_t&
VehicleID, const tServiceInfo&
myService)}
```

Fig. 33. Illustration of the case study SWC communication interface model using FCDL and FIDL.

4.2.1.2 Robotics Domain: ROS Framework IDL: MDL and SDL

ROS (Robot Operating System) provides the required tools to easily access the sensors' data, to process that data, and to generate an appropriate response for the motors and other actuators of the robot. Due to these characteristics, ROS is a perfect framework for self-driving cars and an autonomous vehicle can be considered just as another type of robot [69][13]. ROS offers a message passing interface or IDL model 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 asynchronous nature of publish and subscribe messaging works for most of the Data Distribution Services (DDS) requirements in robotics, however, for specific synchronous request and response interactions, RPC is also used between processes required for higher levels of robot operations. In case of the exchanged information having *data semantics* (using DDS: Data Distribution Services) and being communicated mostly asynchronously (non-blocking mode) between invoker and invoke, this functionality is achieved through introduction of the messages and the concept of *topics* to which the messages are published for subscription. In ROS in case of the exchanged information having *command* semantics and being communicated mostly synchronously (blocking mode) between invoker and invoke, this functionality is achieved through introduction of a service concept. In ROS component models or nodes are described using *Message Description language (MDL)* or *Service Description Language*

(SDL) based on data or command semantics requirements. The *nodes* can only receive messages with a matching *topic* type. An example to create a ROS2 framework service and a corresponding client node for case study on *SeatHeating* SWC using a node handler for invocation of RPC communication, are illustrated Fig. 34.

```
/* Service node created by Server*/
ros::init(
argc,argv,"add_two_ints_server");
ros::NodeHandle n;
ros::SeatHeatingServ service =
n.advertiseService("update_seat_temperatu
-re", update);
```

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

Fig. 34. Illustration of the case study SWC communication interface model using MDL and SDL.

Telematics Domain: Android Application Framework: AIDL

An Android application runs in its own process and cannot access the data of another application running in a different process. To allow one application to communicate with another running in a different process, Android provides an implementation of IPC (Inter Process Communication) through the Android Interface Definition Language (AIDL). It allows to define the programming interface that both the client and service agree upon in order to communicate with each other using IPC. Unlike AUTOSAR Adaptive applications, for most of the Android apps, the service does not need to perform multi-threading, so using a Messenger allows the service to handle one call at a time. If its' important that the service to be multi-threaded, use of AIDL is preferred as a precondition to define the interface[70].

There are a few basic rules one should be aware of when implementing your AIDL interface:

- Incoming calls are not guaranteed to be executed on the main thread, so one needs to think about multithreading from the start and properly build the service to be thread safe.
- By default, RPC calls are synchronous.
- The client binds to service using *IBinder* stub, as illustrated for the *SeatHeating* SWC casestudy in Fig. 35.
- No exceptions that services thrown are sent back to the caller or client.

```
private val binder = object: ISeatHeatingService.Stub() {override fun getPid(): Int =

Process.myPid()}

class SeatHeatingService : Service() {override fun onCreate() {super.onCreate() }

override fun onBind(intent: Intent): IBinder { return binder}

private val binder = object: ISeatHeatingService.Stub() {

 override fun getPid(): Int {

return Process.myPid()

 }val mConnection = object: ServiceConnection {    override fun onServiceConnected(className:
ComponentName, service: IBinder) {       iSeatHeatingService = ISeatHeating.Stub.asInterface(service)    }
```

Fig. 35. Illustration of the case study SWC communication interface model using AIDL.

4.2.1.3 Automotive Domain: AUTOSAR Classic IDL: ARXML

Like AUTOSAR Adaptive AUTOSAR Classic platform (AR CP) also uses ARXML (AUTOSAR XML) as the standard data exchange format. There are three different port prototypes used by the SWCs for interface using AR CP framework. There are several types of port interfaces bound to the port prototypes used in AR CP. For simplicity and to focus more on semantic mapping between IDLs, only Client-Server Interface and Data Interface are considered in current scope. The *ClientServer* interface in AR CP framework is an operation-based interface. There are three types of data passing interfaces used by application SWCs in AR CP framework. These are namely, *SenderReceiverInterface*, *ParameterInterface* and *NvDataInterface*. Fig. 36illustrates *SeatHeating* vehicle application using AR CP platform specific framework ARXML[43][13].

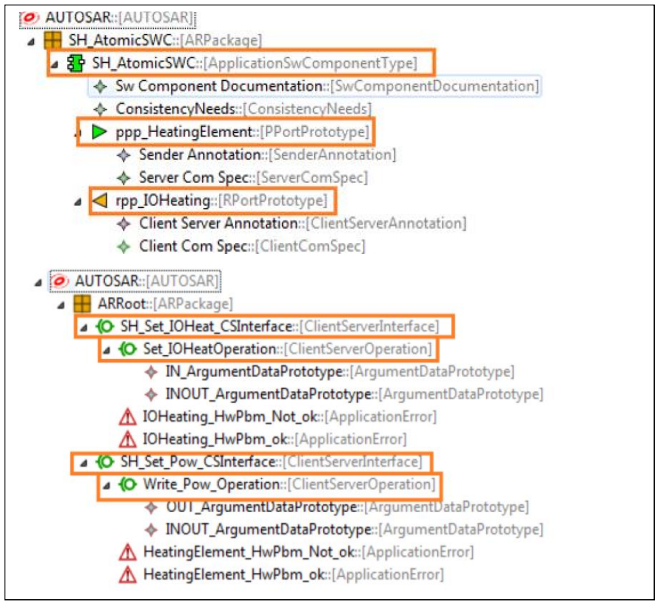


Fig. 36. Illustration of the case study SWC communication interface model using ARXML.

4.2.1.4 Infotainment Domain: Google Protocol Buffers (Protobuf) IDL

Google Protobuf (Protocol buffers) are a flexible, efficient, automated mechanism for serializing structured data, used in IVI systems e.g., vehicle telematics data exchange, etc. 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 such as upgrading of Protobuf from version 2 to 3. Versioning is done by using unique field numbers. The base of defining interfaces within Protobuf is through the definition of messages. However, Protobuf as an IDL is also used by an RPC communication framework known as gRPC (Google RPC) for deployment or interchange of service messages from servers to clients using request and response calls. When using with gRPC, Protobufs are a flexible, efficient, automated mechanism for serializing structured data. The Search Request for *SeatHeating* (case study) service or Search Response messages as per proto 3 syntax is illustrated in Fig. 37. Protobuf messages are identified by a name and contain fields, each with a unique field number. For the transport or deployment of messages as services from skeleton to stub, the Protobuf uses UDP transport protocol for inter host communication and synchronized shared memory for inter process communication [13][31].

```
syntax = "proto 3";
/*Search request for Interface
SeatHeating Operation */
message SeatHeatingOperationRequest {
  String parameter1 = "HeatingElement";
  bool isparameter2 = true;}

/*Search response for Interface
SeatHeating Operation */
message SeatHeatingOperationResponse {
  bool isparameter2 = true;
  String param3 = "HeatingOperation";}

/*gRPC: Bind response to request */
Service SeatHeatingserv {rpc
operation(SeatHeatingOperationRequest);
returns(SeatHeatingOperationResponse);}
```

Fig. 37. Illustration of the case study SWC communication interface model using Protobuf.

Protocol Buffers is a simple language-neutral and platform-neutral IDL for defining data structure and schemas and programming interfaces. It supports both binary and text wire formats and works with many different wire protocols on different platforms. Hadoop Framework uses Protobuf as one alternative in its Serialization Layer.

4.2.1.5 Infotainment Domain: Apache Thrift IDL

Thrift was developed at Facebook as a Framework for implementing cross-language Interfaces to services. Thrift uses an IDL to define the Interfaces and uses an IDL file to generate the *stub*-code to be used in implementing RPC Clients and Servers. The *stub*-code can be used across different languages to access different underlying systems. Thrift has few drawbacks for e.g., it does not support internal compression of data and cannot be splittable. Thrift generates all the necessary code to build RPC clients and servers that communicate seamlessly across various programming languages and is frequently used in the vehicle apps e.g., monitoring of driver behavior apps using sensors [73]. Apache Thrift IDL has forward & backward compatibility. Thrift is robust to version changes. Thrift IDL is a superset of *Protobufs*, with additional features such as constants, rich container types e.g. list, maps, sets, etc. The RPC invocation in thrift is done by sending a method name on the wire as a string. Thrift defines service interfaces using Structures. Thrift is typically used on top of the TCP/IP (Transmission Control Protocol/Internet Protocol) stack with streaming sockets as the base layer of the communication stack. Fig. 38 illustrates *SeatHeating* (case study) using Apache Thrift IDL [73][13].

```
/*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);
}
/*Generated Interface C++ code*/
class SeatHeatingservtf {
public: virtual bool
updatetemp(SeatHeatingElement &elem)=0;}
```

Fig. 38. Illustration of the case study SWC communication interface model using Thrift IDL.

4.2.1.6 Cloud Domain: Open API Specification

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.

```
openapi: 3.0.3
servers:
  # Added by API Auto Mocking Plugin
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/Conti_Auto_Heat_SW
      /Availability/1.0.0
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/Auto_SW/SeatHeating/1.0.0
info:
  description: This is a simple API
  version: "1.0.0"
  title: Simple SeatHeating API
  contact:
    email: you@your-company.com
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
tags:
  - name: SeatHeating
    description: For Monitoring of heating of Car Seats
  - name: UpdateTemperature
    description: Operations to check temperature of car seat
paths:
  /inventory: /getListUpdatedTemperature
    get:
      tags:
        - getUpdatedTemperature
      summary: Give the updated temperature of car seat
      operationId: getupdatedtemperature
```

Fig. 39. Illustration of the case study SWC communication interface model using OpenAPI specification version 3.0.3.

When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. The OpenAPI specification is supported by open-source tool named, *SwaggerHub* . This API specification is used to define the cloud-based services for vehicles such as IoT in vehicle, etc. and supports API formats like YAML and JSON. An OpenAPI document that conforms to the OpenAPI Specification is itself a JSON object, which may be represented either in *JSON* or *YAML* format. The API versions supported by the SwaggerHub tool for API design and documentation are 2.0 and 3.0.0. Fig. 39 illustrates the case study using OpenAPI specification version 3.0.3[74]. Fig. 40 illustrates the basic method calls used for OpenAPI specification for RESTful communication.

Method	Description
GET	Transfer a current representation of the target resource.
HEAD	Same as GET, but only transfer the status line and header section.
POST	Perform resource-specific processing on the request payload.
PUT	Replace all current representations of the target resource with the request payload.
DELETE	Remove all current representations of the target resource.
OPTIONS	Describe the communication options for the target resource.
TRACE	Perform a message loop-back test along the path to the target resource.
PATCH	Apply partial modifications to a resource

Fig. 40. Overview of semantics for all operations using OpenAPI specification.

Each new minor version of the OpenAPI Specification allows any OpenAPI document that is valid against any previous minor version of the Specification, within the same major version, to be updated to the new Specification version with equivalent semantics. Such an update only requires changing the openapi property to the new minor version. Therefore, OpenAPI specifications supports backward compatibility.

4.2.2 Summarized Semantic Comparison of Vehicle Domain IDLs

TABLE III. illustrates the semantical comparison of the various IDL models based on mapping of non-functional traits like *Versioning* i.e., forward or backward compatibility of a components' interface model, software License supported, and *Language bindings* supported i.e. programming languages that can be used to describe an interface model [3]. The framework specific language bindings used to code for components' interface model describes the framework specific interface syntax mapped to a semantic domain[13].

TABLE III. STATIC SEMANTIC MAPPING TABLE BASED ON NON-FUNCTIONAL TRAITS FOR VEHICLE FRAMEWORKS' IDLS

Framework	Version Support	Software License	Programming language support	Binding Comm. Protocol	Interface Description Language
AUTOSAR Adaptive	Backward Compatibility	AUTOSAR	C++	RPC (SOME/IP), REST	ARXML
Franca+	Backward Compatibility	Genivi Alliance	C, C++, java	RPC (SOME/IP)	FIDL, FCDL
Android	Forward and Backward Compatibility	BSD	C, C++, Java	RPC (SOME/IP), REST	AIDL
ROS	Absence of forward & backward compatibility	BSD	C, C++, Python	RPC (SOME/IP), DDS	SDL, MDL
AUTOSAR Classic	Backward Compatibility	AUTOSAR	C	RPC (SOME/IP)	ARXML
Apache Thrift	Forward and Backward compatibility	Apache	C, C++, Java, Python, GO, C#	RPC . can also be customized for REST	Thrift

Google Protocol Buffers	Forward and Backward compatibility	BSD	C, C++, Java, Python, C#	RPC	Protobuf
OpenAPI Specification	Backward Compatibility	Apache	Java, python, C++, Kotlin, Groovy, etc.	REST, AMQP, Apache Kafka	YAML, JSON

TABLE IV. illustrates the semantic mapping of vehicle application IDL alternatives based on functional traits such as interface end points specification or representation, interfaces Communication Design Pattern used for data exchange, and the middleware communication paradigm that is used for deployment of APIs of heterogeneous vehicle frameworks. Functional traits represent the basic structural and behavioral features of a vehicle framework component s' service interface Metamodel[13].

TABLE IV. STATIC SEMANTIC MAPPING TABLE BASED ON FUNCTIONAL TRAITS FOR VEHICLE FRAMEWORKS' IDLS

IDL	Interface end points	Communication Design pattern Used	Communication paradigm
FIDL and FCDL	Ports	Publish-Subscribe, Client-Server	RPC
Protobuf	MessageHandler	Client-Server	RPC
Thrift	MessageHandlers	Client-Server	RPC
ARXML	Ports	Client-Server, Publish-Subscribe	RPC, REST, DDS
MDL and SDL	MessageHandlers	Client-Server, Publish-Subscribe	RPC, DDS
Open API	Messages	Client-Server	REST
AIDL	Messages	Client-Server	REST

TABLE V. . illustrates synergies in semantic traits of communication or middleware protocol stacks that are used as a part of a middleware solution for the deployment of vehicle service-based API models of heterogeneous vehicle frameworks to different targets in order to achieve service-oriented communication. The semantic traits synergies that were also successfully explored to find the commonalities in the communication protocol stacks or middleware solutions, explores the possible scope of reusable functionality of middleware solutions used by heterogeneous vehicle frameworks. These commonalities observed in semantic traits of framework specific communication protocol stacks are considered for the abstraction of framework specific APIs from middleware's solutions for the *Generic API repository* as illustrated in Fig. 29[13].

TABLE V. STATIC SEMANTIC MAPPING TABLE BASED ON COMMUNICATION PROTOCOL USED FOR VEHICLE FRAMEWORKS' IDLS

Middleware/ Communication Protocol Features	Thrift	DDS (Prot-ocol)	SOME /IP	TCPROS
Language bindings supported	*Yes	*Yes	*Yes	*Yes
RPC	*Yes	No	*Yes	*Yes
Publish/Subscri-be method	No	*Yes	*Yes	*Yes
TCP/IP Transport Used	*Yes	*Yes	*Yes	*Yes
Adaption to Hypervisor	*Yes	*Yes	*Yes	*Yes
IDL supported	*Yes	*Yes	*Yes	*Yes

In future, these commonalities explored in middleware communication protocols can also assist more in the direction of cross-platform vehicle application communications, when using a meta-standard domain specific generic interface solution, for deployment of generic APIs to different target frameworks for service-oriented communication, using translation of semantics. With the growing demands for services in future, the functional and non-functional semantic traits along with the IDL alternatives considered in the current scope for vehicle framework IDLs could be further extended, for the semantic analysis in future.

4.3 Technology and Platform Agnostic Specification for Service API Models for Vehicle Domain Heterogeneous SWC Frameworks

Based on the survey of the different service API models of various vehicle domain platforms, it can be inferred that the semantic commonalities and compatibilities between the heterogeneous vehicle platforms SWC frameworks service API models can be effectively simpler to explore and evaluate when the given heterogeneous SWCs' API models can be represented using a unified, standard, programming language-agnostic interface description, such as the OpenAPI Specification (OAS), as seen in Fig. 41[84][85]. When properly defined via OpenAPI, a client API can understand and interact with the service providing SWC API with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, OpenAPI documents describe an APIs services and are represented in either in YAML or JSON formats. It does, however, require the capabilities of the service be described in the structure of the OpenAPI Specification.

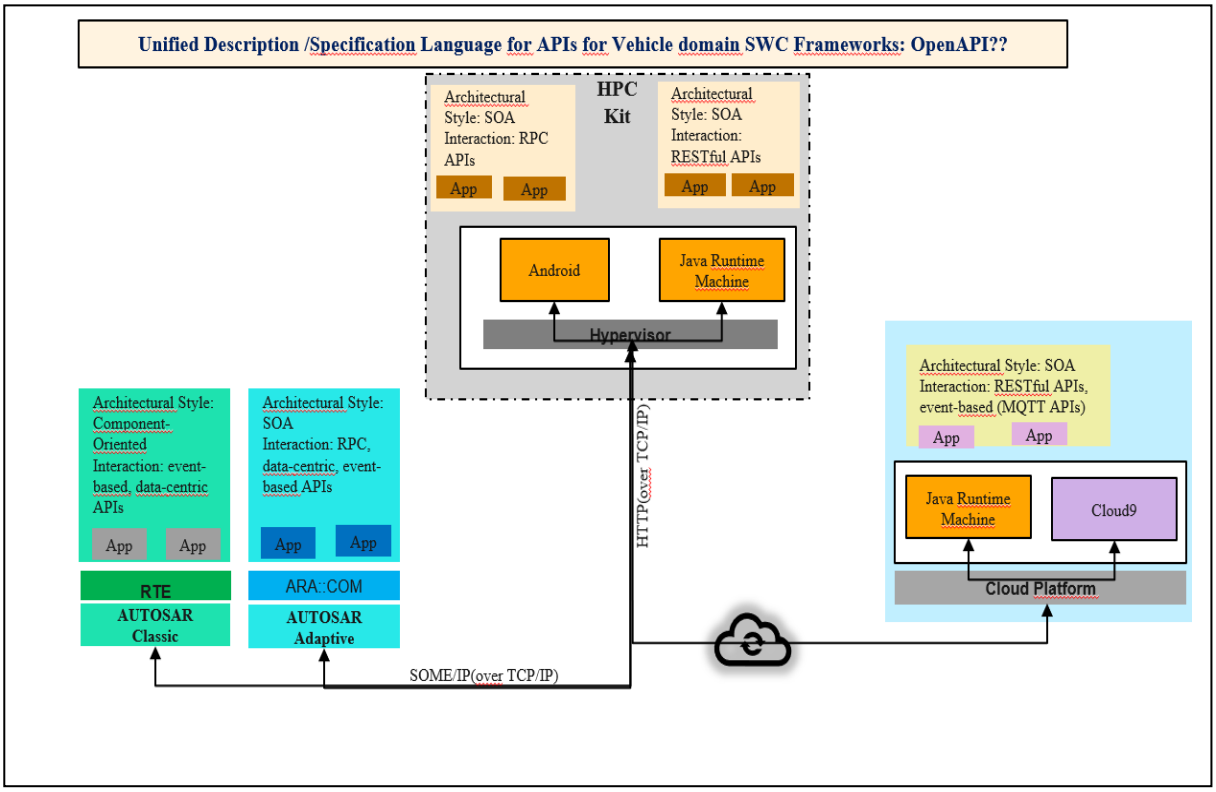


Fig. 41. Overview of cross-enterprise application framework communication for vehicle services.

The OpenAPI Specification (OAS) does not require rewriting the existing API models of heterogeneous vehicle domain platforms. It does, however, require the capabilities of the service be described in the structure of the OAS format. Independent of platforms and technologies, application SWC's API code can be generated in different programming language templates using OpenAPI Codegen tool based on the API specifications that is given in YAML or JSON format [85]. Not all services can be described by OpenAPI – this specification is not intended to cover every possible style of HTTP APIs but does include support for APIs with REST endpoints, as illustrated in Fig. 42. OpenAPIs can be generated using an OpenAPI generator. The framework commonly used for the generation of OAS is *Swagger*. The framework also supports *Codegen* tool to generate code in various programming languages like C, C++, Java, Python, Go, etc. from the OAS [85]. This feature of *Swagger* framework makes OpenAPI as a most suitable candidate for unified API description in context of TABLE III. As observed from TABLE IV. , most of SWC frameworks' IDLs support exchange of messages or information between SWCs' APIs using *Message Handlers*. Within this given context, OAS stands again above other given

IDLs as a suitable candidate for component API specifications. This is due to the fact that OAS uses methods like *GET, POST, PUT, DELETE* for exchange of messages of various formats.

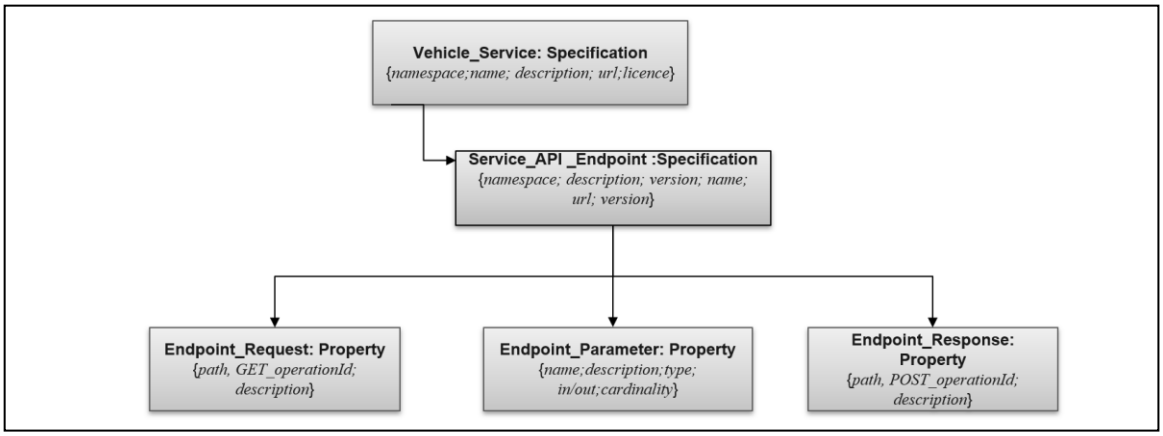


Fig. 42. Work Flow Illustration of vehicle service SWC communication API model using OpenAPI specification (OAS).

4.3.1 Limitations and Solution Proposal

As observed from TABLE IV. SWC frameworks of heterogeneous vehicle platforms relies on RPC-based API communication. This is due to the fact that application software developers have an application that is implemented as multiple distributed components, and those components call each other's APIs for the complete application to function. Therefore, when developers design APIs to solve these kinds of problems, the solution characteristics they will typically prioritize are ease of programming for both the client and the server, and efficiency of execution. RPC is a good match for these priorities. Nevertheless, in order to use the standard OAS for specification of service API models using a unified template, it would be beneficial to expand the scope of the OAS to include the RPC semantics. As suggested, RPC messages could be added in OAS, by adding a few new fields to provide protocol buffer semantics. Since protocol buffers supports RPC communication, properties of a message correspond to fields in a protocol buffer message. In Fig. 39, *x-field-number* (or *fieldNumber*) is a required integer property that associates a unique field number with each property. For illustration on OAS including protocol buffer semantics, let us consider an example on a "Bookstore" API described using RPC based protocol buffer messages, also seen in Fig. 43[86].

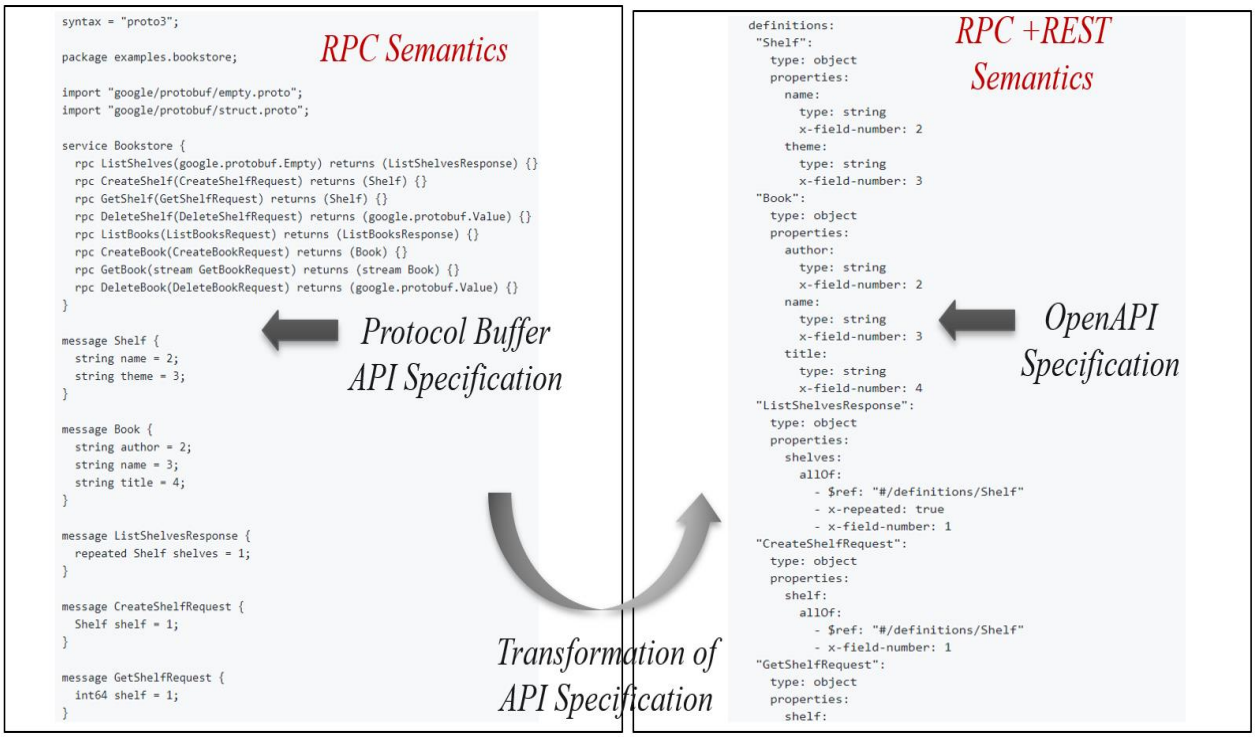


Fig. 43. Representation of RPC communication semantics using OpenAPI specification (OAS).

The OpenAPI specifies "schema objects" which contain information to be used by the underlying libraries to create the HTTP messages, for example "object servers" which defines a part of the URL through which an endpoint is identified. To illustrate more on semantic mapping between RPC architectural semantics and REST architectural semantics, let us consider the semantic mapping between RPC based Franca API specification and REST based OAS in the following table.

TABLE VI. STATIC SEMANTIC MAPPING TABLE BASED ON COMMUNICATION PROTOCOL USED FOR VEHICLE FRAMEWORKS' IDLS

OpenAPI Specification (using HTTP)	**Franca API Specification/FIDL (using RPC)**

GET	**method** Method_Name{
	in{
	<String> Resource_indentifier
	}
	out{
	<**ByteBuffer**> Returned_representation
	}
	error{
	HTTP_error_code_1 = 1
	}
	}
	Description:
	▪ the server publishes the method.
	▪ clients call the method, passing in the "in" argument the resource identifier.
	▪ the server responds with the representation of the target resource and an HTTP error code.

HEAD	**method** Method_Name{
	in{
	<String> Resource_indentifier
	}
	out{
	<**ByteBuffer**> Returned_headers
	}
	error{
	HTTP_error_code_1 = 1
	}
	}
	Description:
	▪ the server publishes the method.
	▪ clients call the method, passing in the "in" argument the resource identifier.
	▪ the server responds with the headers containing the requested meta data and an HTTP error code.

POST	**method** Method_Name{
	in{
	<String> Resource_indentifier
	<**ByteBuffer**> Representation_to_be_posted
	}
	out{
	<**ByteBuffer**> HTTP_specific_info
	}
	error{

	HTTP_error_code_1 = 1 } } *Description:* ▪ the server publishes the method. ▪ clients call the method, passing in the "in" arguments the resource identifier and the representation to be posted. ▪ the server responds with an HTTP error code and possible other HTTP specific information.

PUT	**method** Method_Name{
	in{
	<String> Resource_indentifier
	<**ByteBuffer**> Representation_to_be_updated
	}
	out{
	<**ByteBuffer**> HTTP_specific_info
	}
	error{
	HTTP_error_code_1 = 1
	}
	}
	Description:
	▪ the server publishes the method.
	▪ clients call the method, passing in the "in" arguments the resource identifier and the representation to be updated /replaced by the server.
	the server responds with an HTTP error code and possible other HTTP specific information.

DELETE	**method** Method_Name{
	in{
	<String> Resource_indentifier
	}
	error{
	HTTP_error_code_1 = 1
	}
	}
	Description:
	▪ the server publishes the method.
	▪ clients call the method, passing in the "in" argument the resource identifier of the resource for which the **association** to its current functionality should be removed by the server.
	▪ the server responds with an HTTP error code and possible other HTTP specific information.

However, in context of specification of SWC API models in a human readable format using OAS, one of the daunting impediments is specification of legacy application frameworks like AUTOSAR Classic, AUTOSAR Adaptive, etc. SWC API models using open-source OAS. The on-going work in progress of this research study also focuses on this work direction and try to find resolution to this impediment soon.

Chapter 5 Semantic Comparison of Vehicle Component Frameworks' Interface Metamodels

In MDE models are described by modelling languages, where modelling languages themselves are described by so called meta-modeling languages. A modelling language consists of an abstract syntax, at least one concrete syntax and semantics. Guided by the literature artifacts in Chapter 2, for the illustration of static semantics analysis of vehicle SWC frameworks' interfaces using MDE based approaches, some of the automotive domain heterogeneous platforms based SWC frameworks' communication interface metamodels as *Component-Port-Connector* (CPC) model representation were considered to extract and focus only on the components interface semantic description. Towards simplifying the approach to semantic comparisons of interface fundamental traits and exploring semantic synergies between vehicle domain SWC frameworks' interface metamodels, an abstract, generic, platform-independent, domain-specific SWC interface template was considered as generic domain interface reference *CPC* model.

In general, abstraction of a *CPC* model emphasizes on the common interface semantic properties and hide the computational platform specific details that are not needed in the interface description. In the vehicle domain, from a modeling perspective there are overwhelming number of communication design patterns on how to implement even a simple bidirectional communication using legacy or open source platforms' SWC frameworks' interfaces. Due to the presence of several enterprise platform-specific frameworks' interfaces and their vocabularies of concepts represented by IDLs, results in a source of discord in understanding of the meaning of these concepts by experts from other knowledge domain platforms. Exploration of interface semantic synergies is essential for effective collaboration of services through semantic interoperability between various application SWC frameworks within the umbrella of automotive domain to support complex and novel service requirements of the new automotive era. The vehicle domain interface reference *CPC* model includes general and platform-independent interface traits.

To understand the abstraction of interface semantic traits for the generic domain interference reference *CPC* model, let us consider for e.g. an assembly or a composition of software components *A* consisting of two different components *C1* and *C2*. *P1* and *P2* denotes specific interface semantic traits of the components *C1* and *C2* as seen in Fig. 44. The abstract interface semantic traits of software composition *A* that is represented as P_A, can be predicted based on *P1* and *P2*. From the Fig. 44, P_A can be considered to symbolize the fundamental semantic interface traits of the given generic domain interface reference *CPC* model, that are abstract and can be predicted based on semantic traits of various heterogeneous vehicle application framework components' interfaces.

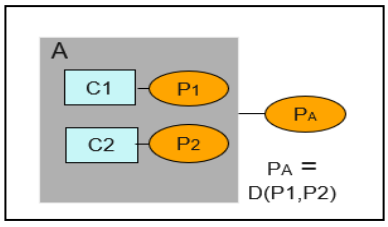


Fig. 44. Abstraction of interface semantic traits for a SWC composition

The structure of an abstract generic domain reference *CPC* model basically illustrates the abstract view of the components' interface types, their typed input and output ports, and the connectors between them. An abstract, generic domain-specific *CPC* model can further provide knowledge base for future implementation of automotive domain specific meta-standards software solution such as coherent, meta-meta-model or a Meta-IDL model, as seen in Fig. 31[75][71].

Considering the abstract, domain-specific *CPC* model as a reference, some of the most used vehicle domain platform-specific application frameworks' SWC communication interfaces were represented as *CPC* models[71]. With the semantic analysis of different framework components *CPC* models, the areas of synergies between service-based interfaces and areas of conflicts were revealed from a functional perspective. In the Fig. 31, a M2 layered based SWC meta-model, namely, *SeatHeating* SWC is considered as a typical example to illustrate abstraction of a SWCs' *CPC* model from a real system model at layer M0.

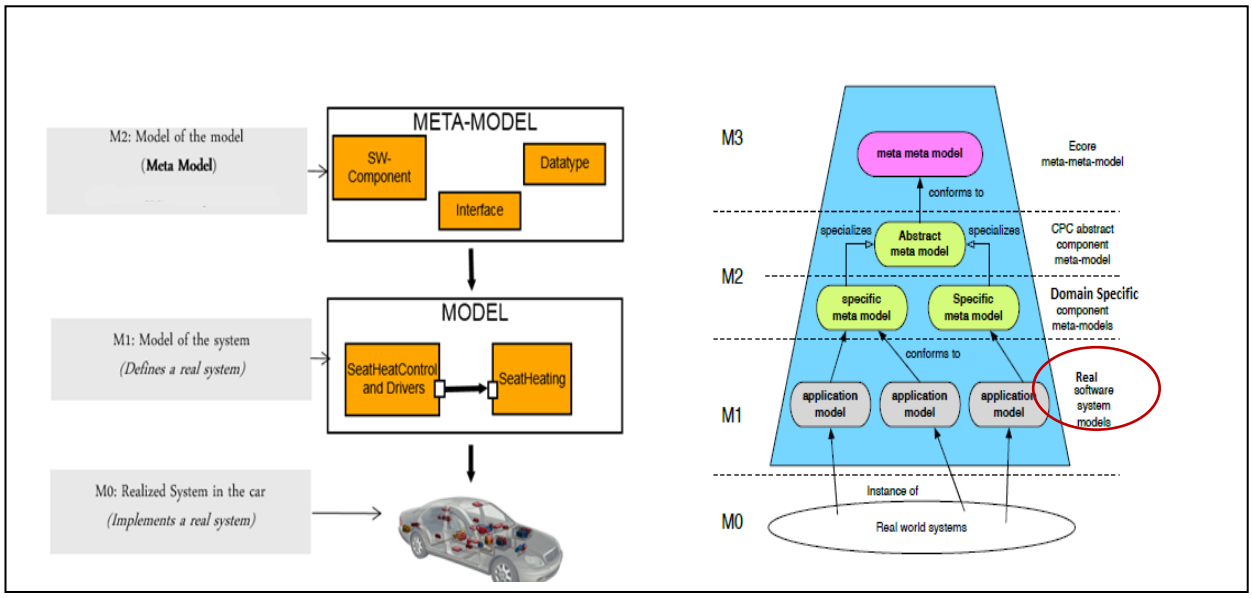


Fig. 45. Illustration of abstraction of standardized four-layered modeling architecture for vehicle domain application SWC .

Fig. 46 illustrates the platform-independent, abstract, domain-specific generic interface reference *CPC* model architecture which is a generic representation of M2 layer based SWC interface metamodel for vehicle domain[75].

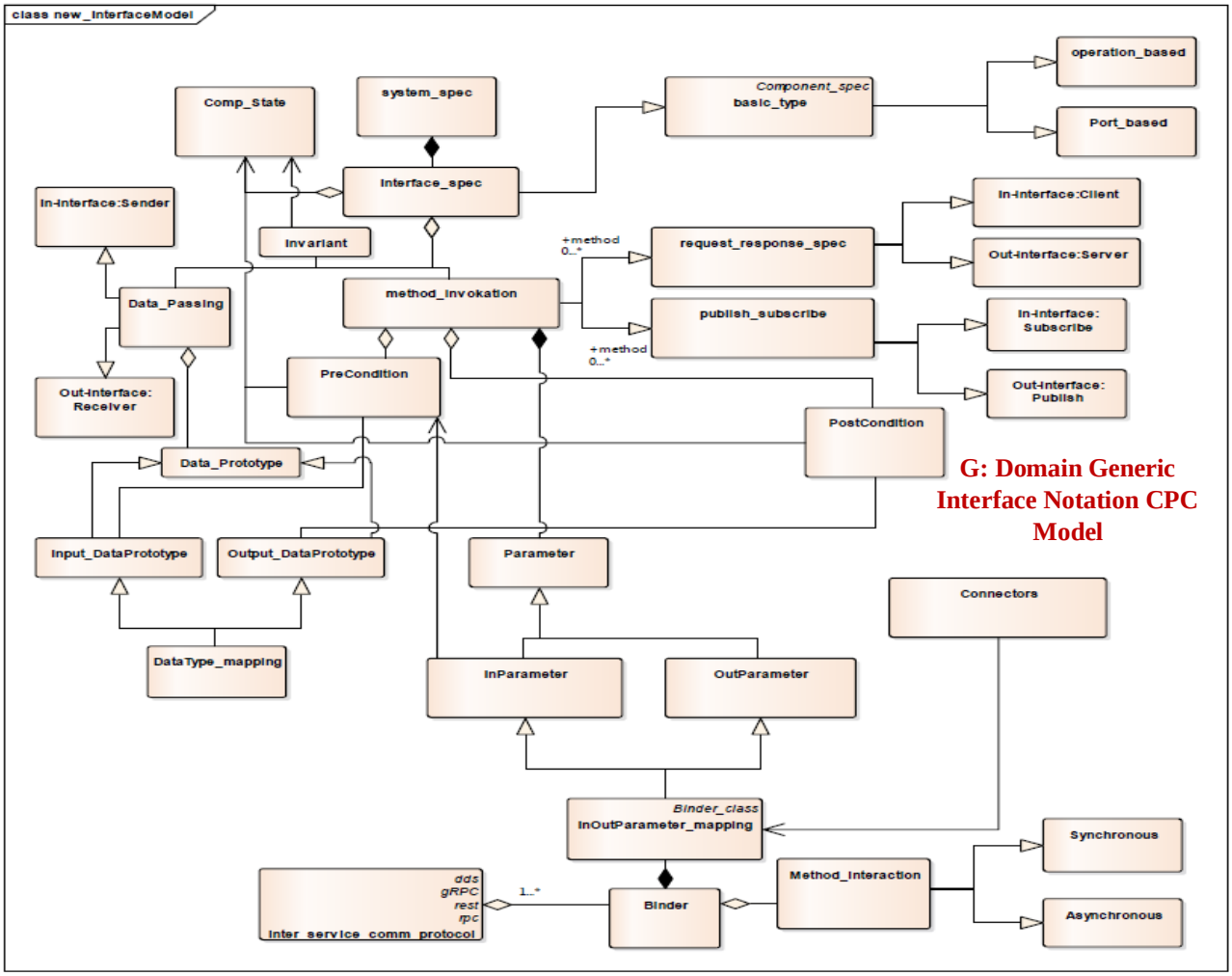


Fig. 46. Abstract representation of platform-independent, generic Component-Port-Connector (CPC) model for vehicle domain application SWC interface.

5.1 Application Component Framework Interface Metamodels: Alternatives

Considering todays' car complex architecture being clustered into different knowledge domains such as infotainment, automotive, robotics, connectivity (telematics), etc. exploring possible semantic synergies in interface concepts between the SWC frameworks of these different knowledge domains can enhance semantic interoperability and chances of service collaborations among them. Following the principles of MDE, the generic *CPC* model (in Fig. 40) is considered as a reference model for representation of the heterogeneous vehicle application frameworks' SWC metamodels from communication interface perspective.

In reference to the interface semantic traits of the domain-specific, reference generic *CPC* model, this Chapter conducts an analysis of semantics traits for each of the cross-enterprise vehicle application SWC frameworks' interfaces' metamodels and identifies the possible areas of interface semantic concepts synergies in order to weave the semantic interoperability between the component models for any future domain specific software solutions or evolution of software meta-standards for SWC frameworks' interfaces. *CPC* metamodel representation for various SWC frameworks abstracts away those platform-specific details that are not required for the component's communication interface description, so that the application developers can focus more on the interface concepts to identify semantic synergies in functionalities[75].

5.1.1 Automotive Knowledge Domain: AUTOSAR Adaptive SWC Framework

Autosar is widely accepted as the de-facto standard of automotive system software architecture for developing automotive application of various automotive platforms during the different phases of a vehicle life cycle. The Autosar Adaptive software component has a service provider port (*PPortPrototype*) and a receiver port (*RPortPrototype*). Each *PortPrototype* is typed using service interfaces. An example of Autosar Adaptive (release version 18-10) framework specific UML profile representation of SWC interface metamodel (at M2 modeling layer) can be seen in the Fig. 47. The service based SWC's interface model employed for interface description is specified using various elements, this includes [41][8]:

- Aggregation of variable data prototypes in the role of Events (*VariableDataPrototype*);
- Aggregation of Getter, Setter and Notifiers in the role of Fields. A *Field* is a combination of a Remote Procedure Call (RPC) and an event.
- Aggregation of *ClientServerOperations* in the role of Methods. Arguments data required for Client-Server Operation is represented in the role of *ArgumentDataPrototype* in the meta-model as seen in Fig. [5] . Method invocation in Autosar Adaptive can be synchronous or asynchronous.

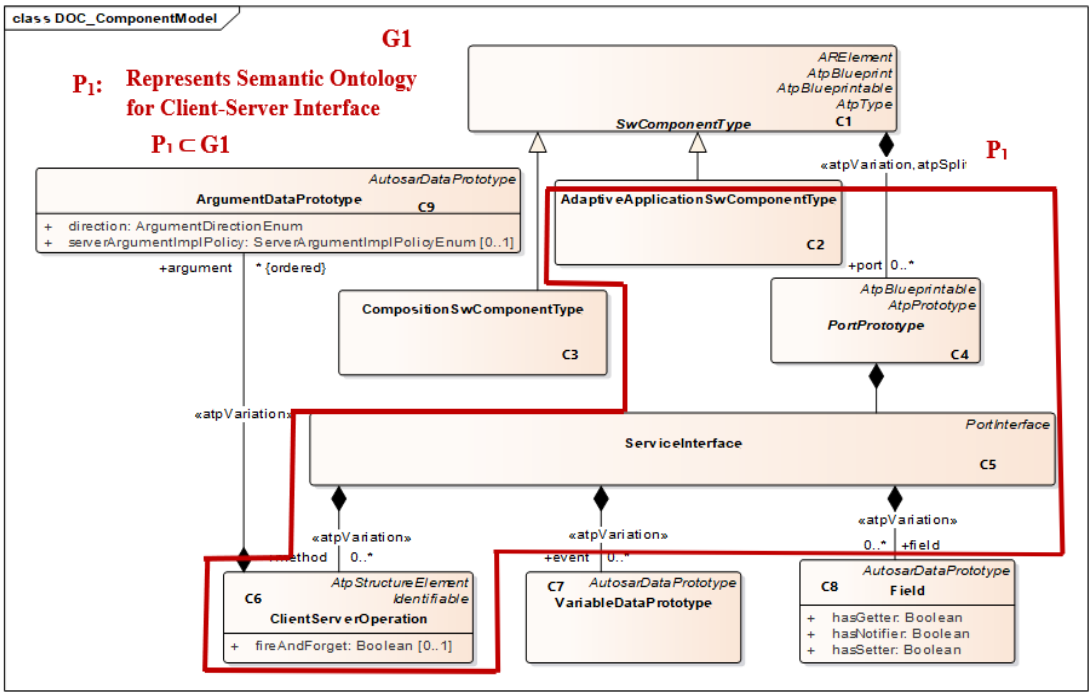


Fig. 47. Graphical model representation (G1) of AUTOSAR Adaptive framework SWC metamodels' constructs for interface.

TABLE VII. illustrates semantic mapping between metamodel constructs of AUTOSAR Adaptive SWC *CPC* model (also represented by abbreviation "C") and the metamodel constructs of abstract generic *CPC* model. The *Methods*, *Events*, *Service Interface* and *PortPrototypes* of AUTOSAR Adaptive framework can be semantically mapped to *method_invocation*,*Data_Prototype*,*Interface_spec* and *Port_based* of the generic *CPC* model[41].

TABLE VII. SEMANTIC MAPPING OF METAMODEL CONSTRUCTS FROM AUTOSAR ADAPTIVE SWC CPC TO GENERIC CPC MODEL

Domain Generic Notation CPC Model		AUTOSAR Adaptive Application Framework CPC Model
method_Invokation	⇐	ARAP:ClientServerOperation (C6)
interface_spec	⇐	ARAP:ServiceInterface (C5)
Port_based	⇐	ARAP:PortPrototype (C4)
Data_prototype	⇐	ARAP:VariableDataPrototype (C7)

5.1.2 Infotainment Knowledge Domain: Franca (including Franca+) SWC Framework

Franca application framework is developed as a part of the GENIVI standard Franca (version 0.13.0) framework and supports IVI (In-Vehicle infotainment) systems' interfaces. Franca+ provides an extension to the native Franca framework that adds support to the modeling of components, composition of components, typed ports (provides and required), Port interfaces (optional major and minor versions) and connectors between ports as seen in the meta-model represented by UML profile in the Fig. 42 [42]. Like AUTOSAR Adaptive, Franca+ framework also supports the *CompositionComponentPrototype* (named as Component). A component contained in a composition is called *ComponentPrototype*. The *service* attribute marks a component as service running on the target platform. For Interface specification at application software component level, Franca uses *Methods*, *Broadcasts* and *Attributes*, as illustrated in the Fig. 42 [8]

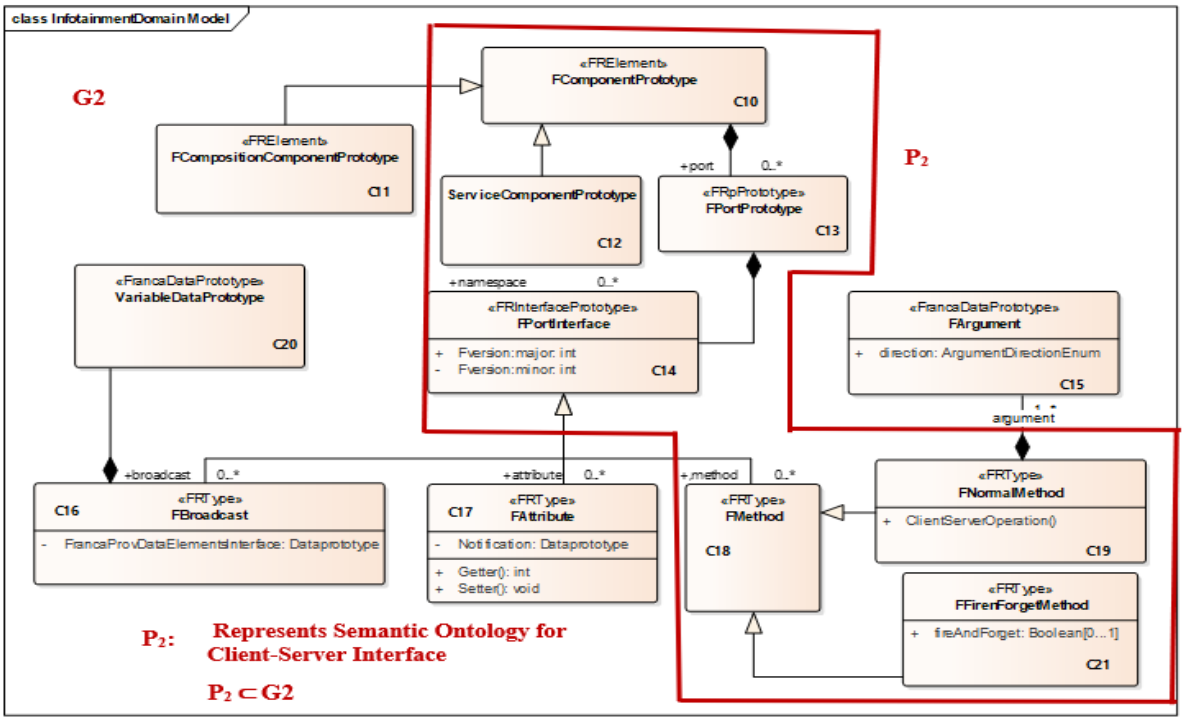


Fig. 48. Graphical model representation (G2) of Franca framework SWC metamodels' constructs for interface.

TABLE VIII. also illustrates semantic mapping between metamodel constructs of Franca+ SWC *CPC* model (also represented by abbreviation "C") and the metamodel constructs of abstract generic *CPC* model. The *FPortinterface*, *FNormalMethod*, *FFirenForegetMethod* and *VariableDataPrototype*, of Franca (also Franca+)

can be semantically mapped to *interface_spec*, *request_response_spec*,*publish_subscribe* and *Data_Prototype* of the generic *CPC* model.

TABLE VIII. SEMANTIC MAPPING OF METAMODEL CONSTRUCTS FROM FRANCA SWC CPC TO GENERIC CPC MODEL

Domain Generic Notation CPC Model	Franca(also Franca+) Application Framework CPC Model
Interface_spec	FR:FPortinterface (C14)
request_response_spec	FR:FNormalMethod (C19)
publish_subscribe	FR:FireForgetMethod (C21)
Data_prototype	FR:VariableDataPrototype (C20)
method_Invokation	FR:FMethod (C18)

5.1.3 Robotics Knowledge Domain: ROS2 Application Framework

The Robot Operating System (ROS) developed by WillowGarage aims to provide a software development environment for robotics. ROS is a perfect framework for autonomous driving cars and provides high-level functions such as route planning, connectivity, etc. Literally, ROS (version 2.0) is not a component-oriented software. However, like in many programming paradigms (objects in object-orientation, etc.), ROS also strives to build apps from modular units. In the ROS programming model, the modular programming unit is a node. *Nodes* are semantically similar to *SwComponentPrototype* in AUTOSAR Adaptive, and *ComponentPrototype* in Franca as can be seen in Fig. 49. A *Topic* can be considered as a named communication channel which is used to send and receive messages between nodes and can be semantically compared to *PortInterface* of Autosar Adaptive and Franca application frameworks [71][30]. There are no software connectors used between the *Nodes* in ROS framework.

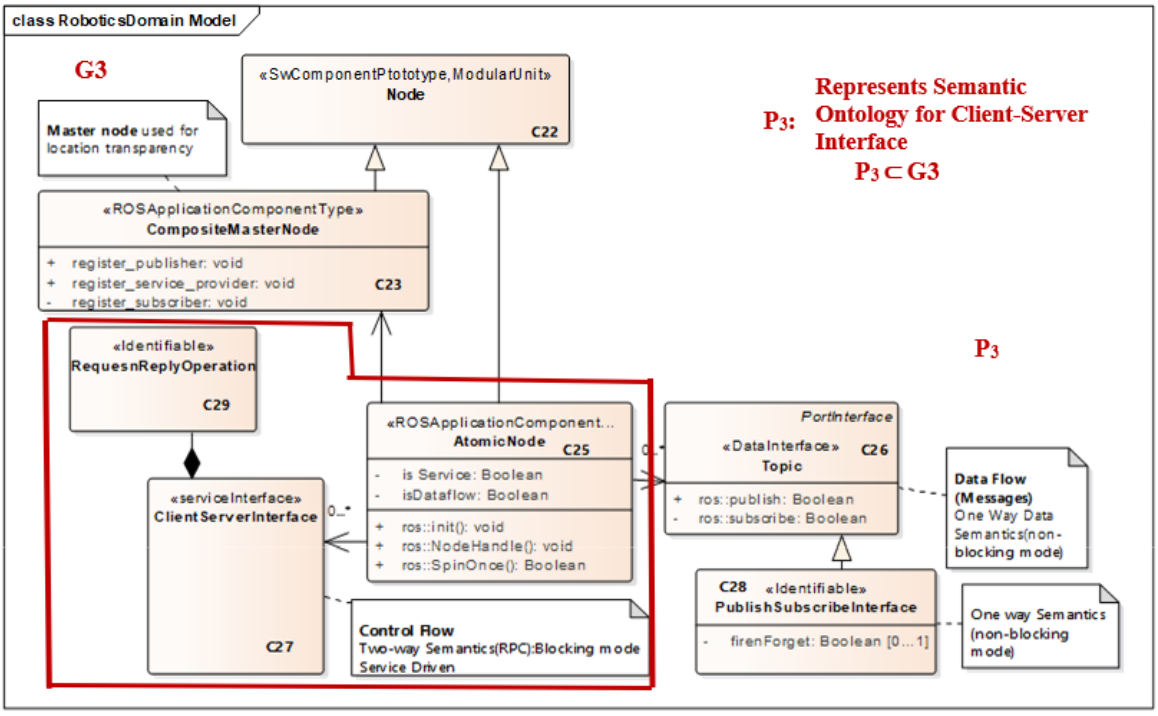


Fig. 49. Graphical model representation (G3) of ROS2 framework SWC metamodels' constructs for interface.

In ROS all the necessary information exchange among nodes is performed through messages. ROS has two basic types of interaction endpoints attached to a node, namely *data* and *command* (or *service*) interface. In case of the exchanged information having data semantics using DDS (Data Distribution Services) and being communicated mostly asynchronously (non-blocking mode) between invoker and invoke like *PublicSubscribeInterface*

communication pattern, this functionality is achieved through introduction of the messages and the concept of *topics* to which the messages are published for subscription. The Data Semantics are semantically similar to the asynchronous *fireAndForget* method invocation of Autosar Adaptive and *FFireForgetMethod* of Franca framework. In ROS in case of the exchanged information having command semantics and being communicated mostly synchronously (blocking mode) between invoker and invoke, this functionality is achieved through introduction of a *service* concept [[8][71]].

TABLE IX. also illustrates semantic mapping between metamodel constructs of ROS2 node *CPC* model (also represented by abbreviation "C") and the metamodel constructs of abstract generic *CPC* model. The *Topics*, *RequestReplyOperation*, *ClientServerInterface* and *PublishSubscribeInterface* of ROS2 can be semantically mapped to *Data_Passing*, *request_response_spec*, *interface_spec* and *publish_subscribe* of the generic *CPC* model.

TABLE IX. SEMANTIC MAPPING OF METAMODEL CONSTRUCTS FROM ROS2 FRAMEWORK SWC CPC TO GENERIC CPC MODEL

Domain Generic Notation CPC Model	ROS2 Application Framework CPC Model
Data_Passing	ROS2:Topic (C26)
request_response_spec	ROS2:ClientServerInterface (C27)
publish_subscribe	ROS2:PublishSubscribeInterface (C28)
method_Invokation	ROS2:RequestReplyOperation (C29)
interface_spec	ROS2:Topic (C26), ROS2:ClientServerInterface (C27)

5.1.4 Connectivity -Telematics Knowledge Domain: Android SWC Framework

Android application frameworks by Google are standard frameworks in Telematics domain, which are widely used to support vehicle services on connectivity. Four different types of application SWCs are used as essential building blocks of an Android application namely, *Activities*, *Services*, *Broadcast receivers* and *Content providers*. Three of the four component types *activities*, *services*, and *broadcast receivers* are activated by an asynchronous message called an *Intent* as seen in Fig. 50. Fig. 50 which represents the SWC *CPC* metamodel from interface perspective, illustrates *startService()* service method call invoked by a client result in a corresponding call to the server or services' *Service.onStartCommand ()* method. On successful service connection binding with the stub or server, the client or *ClientbindingClass* receives an instance of *IBinder* interface using *onServiceConnected()* callback method [13][31]. TABLE X. also illustrates semantic mapping between metamodel constructs of Android SWC *CPC* model (also represented by abbreviation "C") and the metamodel constructs of generic *CPC* model. The *IBinder*, *ExampleContentProvider*, Service and *ClientBindingClass* of Android can be semantically mapped to *interface_spec*,*Data_Passing*,*In-interface:Client* and *Out-Interface:Server* of the generic *CPC* model.

TABLE X. SEMANTIC MAPPING OF META-MODEL ENTITIES FROM ANDROID FRAMEWORK SWC CONSTRUCT TO GENERIC CPC MODEL

Domain Generic Notation CPC Model	Android Application Framework CPC Model
interface_spec	ANDR:IBinder (C34)
Data_Passing	ANDR:ContentResolver (C33)
Out-Interface:Server	ANDR:Service (C31)
In-Interface:Client	ANDR:ClientBindingClass (C37)
method_Invokation	ANDR:OnBind (C31)
Data_Prototype	ANDR:ContentProvider (C33)

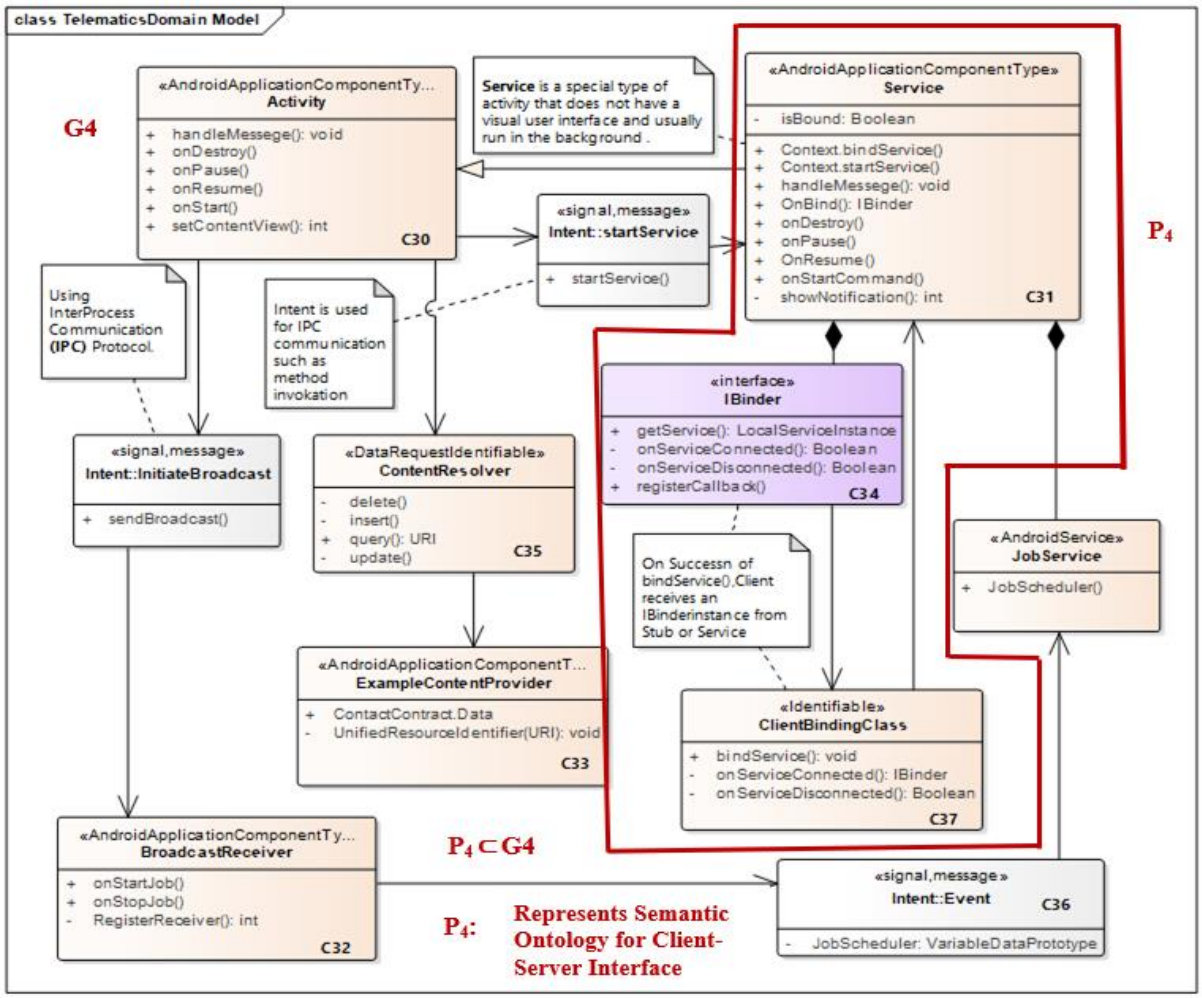


Fig. 50. Graphical model representation (G4) of ROS2 framework SWC metamodels' constructs for interface.

5.1.5 Automotive Domain: AUTOSAR Classic Framework

Autosar Classic application framework is part of AUTOSAR de-facto automotive standard and is used to support deep embedded vehicle features throughout the vehicle life cycle. Due to commonly used ARXML (Autosar XML) as modelling language, possible mappings between metamodels of different AUTOSAR standards can be easily achieved. There are three different port prototypes namely, *Provided*, *Required* and *ProvidedRequiredPortPrototypes* represented as seen in the SWC CPC metamodel in Fig. Each port prototype is typed by a port interface. The Port interface describes the static structure of information exchange as seen in Fig. 51. The *ClientServer* interface in AUTOSAR Classic is an operation-based interface [43].

The *ClientServerOperation* logical functionality can be semantically mapped to *Methods* used by *ServiceInterface* in AUTOSAR Adaptive, *FNormalMethod* in Franca and *RequestReplyOperation* in ROS2. The *ArgumenentDataPrototype* that are used as arguments for *ClientServerOperation* in Autosar Classic can be semantically mapped to *ArgumentDataPrototype* contained within a *Method* as a part of *ServiceInterface* in AUTOSAR Adaptive as seen in Fig. 51. There are also three types data passing interfaces used by application SWCs in Autosar Classic. These are namely, *SenderReceiverInterface*, *ParameterInterface* and *NvDataInterface*. The *SenderReceiver* interfaces passes data of *VariableDataProtype* from *Sender* to the *Receiver*. The *DataElements* of *SenderReceiverInterface* in AUTOSAR Classic can be semantically mapped to *Events* (also of type *VariableDataPrototype*) that are contained within a *ServiceInterface* in Autosar Adaptive, *Broadcast* in Franca and *Topics* in ROS application frameworks[43][8].

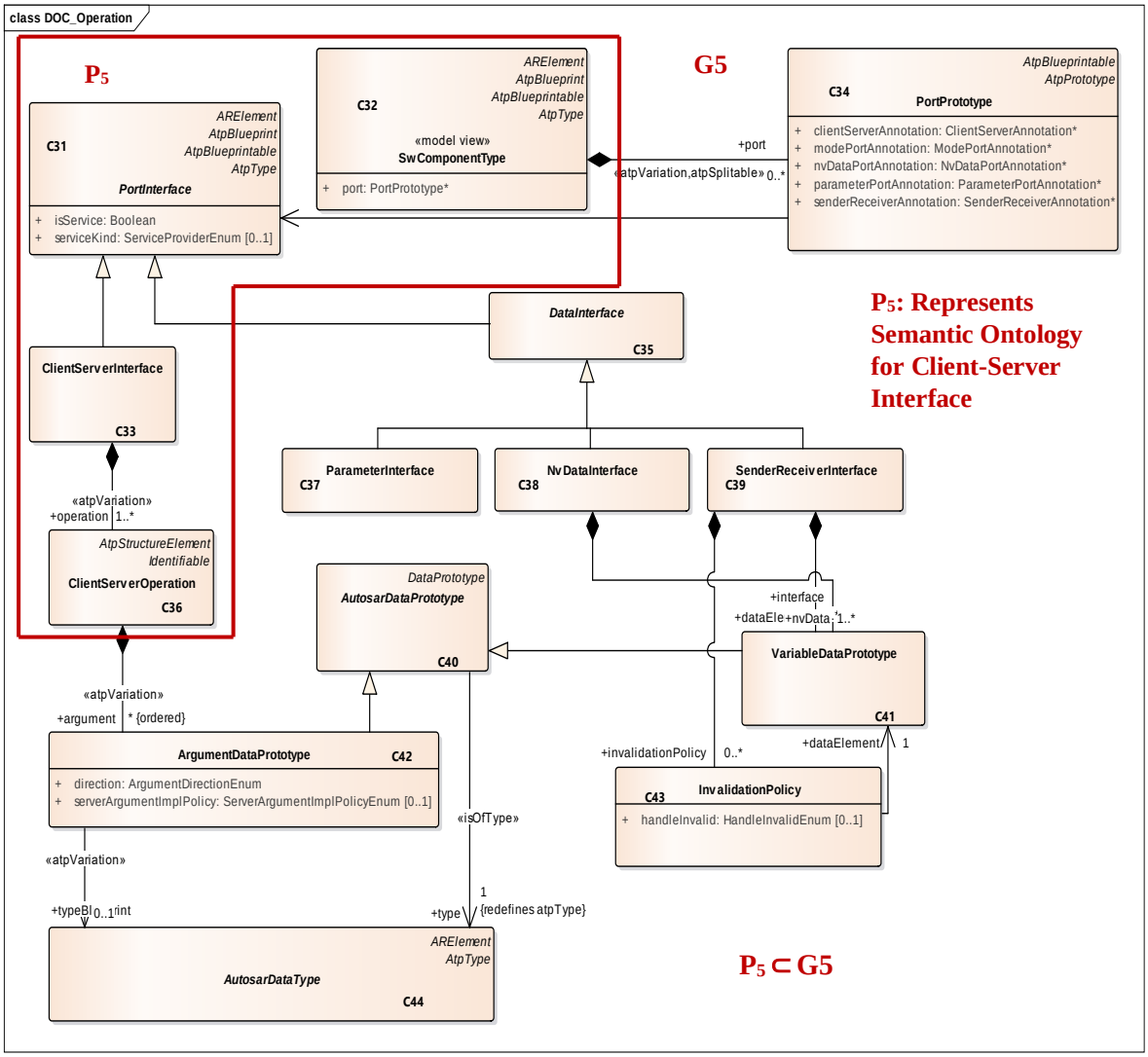


Fig. 51. Graphical model representation (G5) of AUTOSAR Classic framework SWC metamodels' constructs for interface.

TABLE XI. also illustrates semantic mapping between metamodel constructs of AUTOSAR Classic SWC *CPC* model (also represented by abbreviation "C") and the metamodel constructs of generic *CPC* model. The *PortInterface*, *ClientServerOperation*, *DataInterface*, *VariableDataPrototype* and *InvalidationPolicy* of AR CP can be semantically mapped to *interface_spec*, *method_invokation*, *Data_Passing*, *Data_Prototype* and *PreCondition* the generic *CPC* model.

TABLE XI. SEMANTIC MAPPING OF META-MODEL ENTITIES FROM AUTOSAR CLASSIC FRAMEWORK SWC CONSTRUCT TO GENERIC CPC MODEL

Domain Generic Notation CPC Model	AUTOSAR Classic application Framework CPC Model
interface_spec	ARCP: PortInterface (C31)
method_Invokation	ARCP: ClientServerOperation (C36)
Data_passing	ARCP: DataInterface (C35)
Data_Prototype	ARCP: VariableDataPrototype (C41)
PreCondition	ARCP: InvalidationPolicy (C43)
request_responese_spec	ARCP: ClientServerInterface (C33)

5.2 Summarized Semantic Mapping of Component Frameworks Interface Metamodels

The summary of semantic mapping between the interface concepts of the SWCs' *CPC* metamodels of heterogeneous vehicle application frameworks into the generic platform independent vehicle domain *CPC* metamodel (in the Fig.) can be seen in TABLE XII. The objective of the summary table is to identify interface conceptual synergies between the given vehicle application SWC frameworks *CPC* metamodels when using the same domain-specific, generic *CPC* metamodel as a common reference for representation. For the semantic mapping of vehicle application SWC frameworks, the proposed mapping approach attempts to identify the synergies in interface concepts of heterogeneous SWC metamodels, highlighting (represented by letter "*P*" in figures representing SWC metamodels) a common SWC communication interface semantic relation, namely, *Client-Server*, used in all the given application SWC frameworks. TABLE XII. also illustrates information in this regard. The goal for framing the following table is also towards finding a solution for semantically correlating the heterogeneous cross-enterprise platform-specific SWC frameworks and exploring more possibilities towards design of a concrete future automotive domain specific interface meta-metamodel or a MOF[8].

TABLE XII. SUMMARIZED MAPPING TABLE FOR SEMANTIC MAPPING OF META-MODEL CONSTRUCTS FROM HETREOGENEOUS FRAMEWORK SWC CPC TO GENERIC CPC MODEL

Generic CPC Model	Autosar Adaptive CPC Model	Franca (and Franca+) CPC Model	ROS2 CPC Model	Autosar Classic CPC Model	Android CPC Model
Interface_spec	ARAP: ServiceInterface	FR: FPort interface	ROS2: Topic, ROS2: ClientServer Interface	ARCP: PortInterface	ANDR: IBinder
method_invokation	ARAP: ClientServer Operation	FR: FMethod	ROS2: RequestReply Operation	ARCP: ClientServer Operation	ANDR: OnBind
Data_Prototype	ARAP: VariableData Pr-ototype	FR: VariableData P-rototype	Not Available	ARCP: VariableDataPrototype	ANDR: ContentProvider
Data_passing	Not Available	Not Available	ROS2: Topic	ARCP: DataInterface	ANDR: ContentResolver
request_responese_spec	ARAP: ServiceInterface	FR: FNormalMeth-od	ROS2: ClientServerInter-face	ARCP: ClientServerIn-terface	ANDR: Service, ANDR: Client Binding Class
Publish_subscribe_spec	Not Available	Not Available	ROS2: PublishSubscribe Interface	Not Available	Not Available

Part III Design & Implementation Level (WIP)

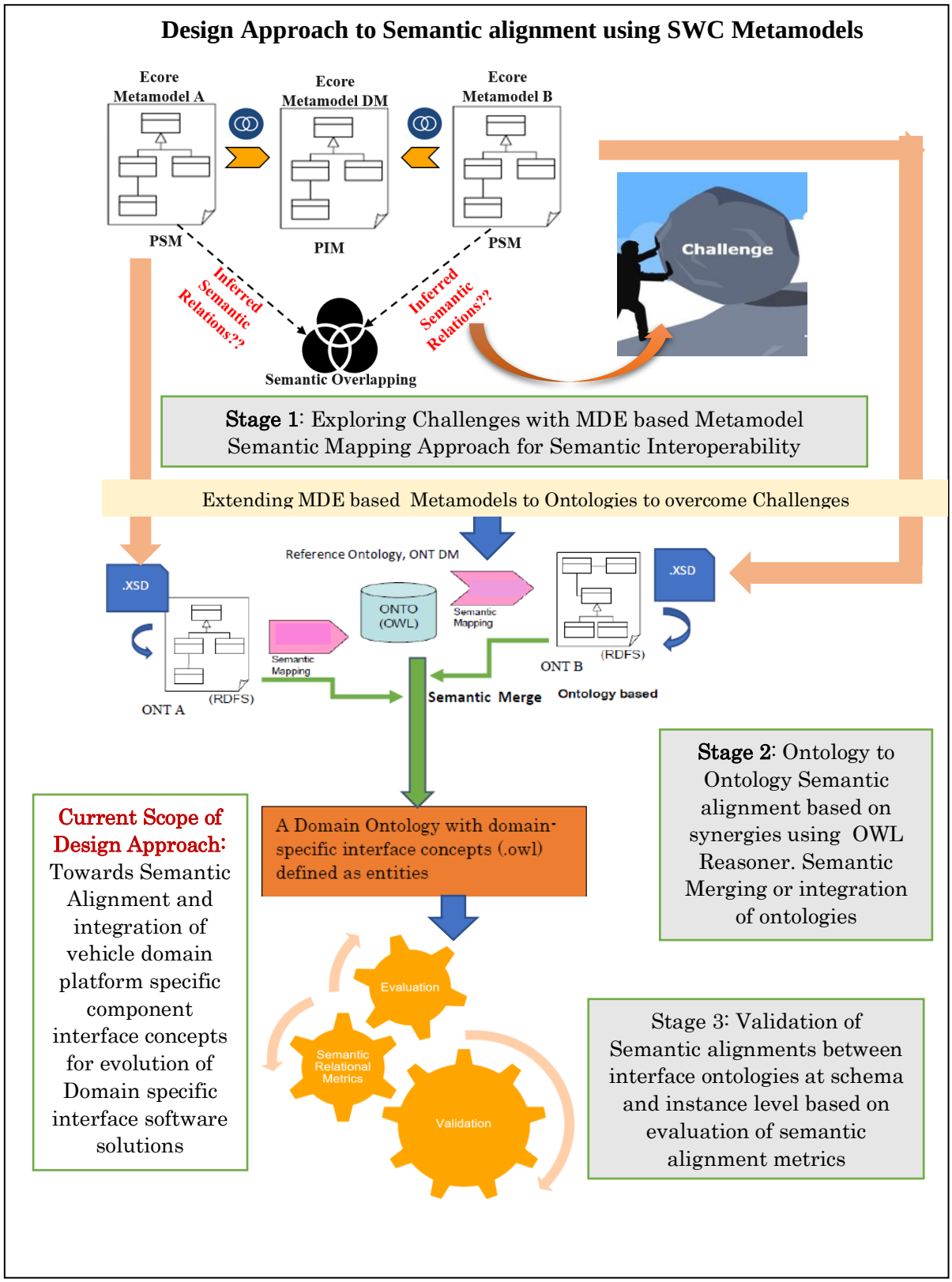


Fig. 52. Illustration of Design Approach to semantic alignment of cross-enterprise vehicle SWC frameworks' interface metamodels.

The design level describes the analysis level design by taking the software component framework interface resources into consideration. This level emphasizes on improvement of design approaches for interface semantic concepts alignment between platform-specific component frameworks within the vehicle domain. Design level constitutes component interface models described using conceptual metamodeling mechanism. In this level, in context of domain analysis of interface models and evolution of an abstract domain-specific global interface solution using semantic mapping and integration, different approaches based on MDE paradigm like ecore metamodeling and ontology paradigm based OWL2 metamodeling are presented. Towards component interface semantic interoperability, these approaches stand to be complementary to one another and not alternative, hence, the proposed methodologies at design level proposes the optimal way to amalgamate both the paradigm approaches in a value-added way.

The extension of the proposed design approach towards implementation is not yet complete and is under progress. However, in context of implementation of a unified vehicle domain interface solution using semantic integration of platform-specific interface ontology sources, the following question arises:

What must be the building blocks of the solution for the semantic integration of interface ontology sources towards evolution of such Domain-specific interface solution?

The OWL2 metamodel classes that implements platform-specific vehicle SWCs' communication interface model concepts must be semantically aligned for the semantic integration by using a shared vocabulary of domain conceptualizations. More important for the interface concepts semantic alignment, a domain-specific, platform-independent, interface metamodel containing only domain-specific generic interface concepts must be used as an intermediate agent to ease the semantic alignment between the platform-specific interface models.

Chapter 6 Design Approach to Semantic Alignment of Component Frameworks Interface Meta-Models

There are numerous ways to tackle semantic interoperability between application SWC frameworks' API models within the vehicle domain. However, due to the wide usage of MDE based modeling tools like Eclipse in automotive domain, for API model design and development, MDE paradigm-based modeling approach was selected as a default modeling approach to tackle semantic interoperability. Therefore, this chapter includes analysis of MDE based modeling approaches, abstractions, and techniques for semantic interoperability of heterogeneous frameworks SWCs' API models. MDE claims that by using PIMs (Platform Independent Models) that abstract away platform-specific details, integration and interoperability across various platforms should become easier to produce [4]. Moreover, it is proven by literature artifacts in semantic web domain, that the semantic alignment and interoperability between heterogeneous platform-specific component frameworks interface models within a domain can be made effectively easier by using an abstract domain specific mediator or reference model as a common, shared agent between the semantically compared model resources [12].

A Mediator for Semantic Interoperability between Interface Metamodel Resources

Mediators connect possibly heterogeneous entities of an interface or API description model which have structural, semantic, or conceptual incompatibilities. They aim at automatically handling interoperability problems between elements of different SWC frameworks' interface models. Considering the syntactic or structural heterogeneity between semantically equivalent API architectures of heterogeneous application frameworks, both MDE and ontology paradigms based semantic alignment conceptual modeling approaches propose to use a common, domain specific reference metamodel, namely DM that contains a set of domain-specific, platform-agnostic generic interface semantic traits that are common to most of the semantically compared interface resources within the domain. DM mediator or reference metamodel can be considered as a shared, intermediate agent to enable semantic mapping between the interface metamodel resources [12][8]. The generic, platform-agnostic, interface semantic traits that are included in the DM mediator metamodel are basically the most common domain specific interface semantic traits observed from the analysis of TABLE III. ,TABLE IV. ,TABLE V. and TABLE XII.

A Vehicle Domain Case Study for Analysis of Semantic Mapping Approaches

To simplify the demonstration of interface metamodel semantic alignment approaches, a common vehicle domain case study was proposed. The case study considered is *Autonomous Maneuvering* from ADAS (Advance Driver Assistance System) functional domain. *Autonomous Maneuvering* controls the decision of an autonomous car whether the collision is avoidable because it determines the type of action the vehicle should automatically take [17]. This case study demands collaborations between various heterogeneous, cross-enterprise vehicle applications from different knowledge domains like automotive, robotics, infotainment, etc. to accomplish its services requirements, as seen in Fig. 53. The case study considers interface models of three most used vehicle domain application component frameworks from heterogeneous knowledge domains such as powertrain, infotainment and autonomous driving for semantic mapping. The three selected vehicle domain application component frameworks are namely, AUTOSAR Adaptive, Franca (from Genivi) and ROS2. The interface models of these three frameworks are accordingly named as Source 1, Source 2 and Source 3 for semantic mapping.

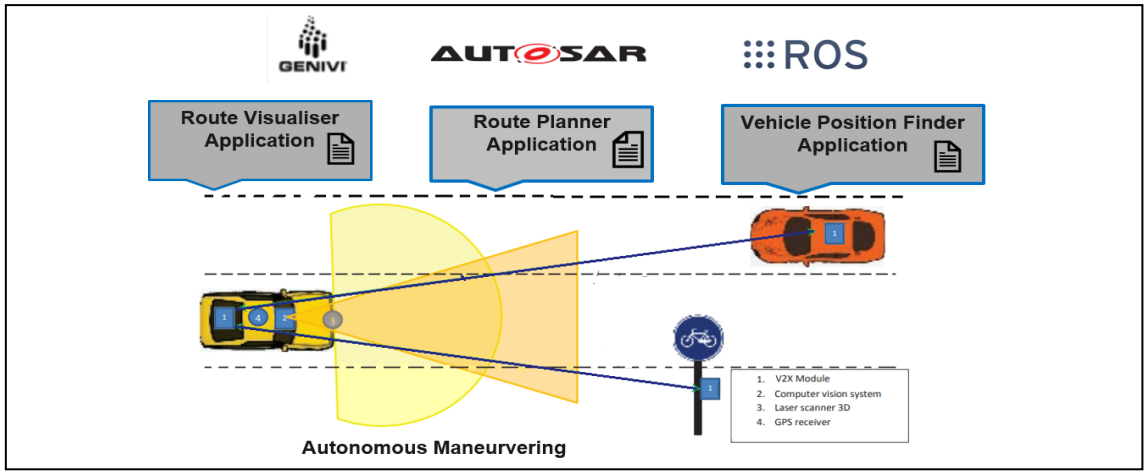


Fig. 53. Autonomous Maneurvering case study involving cross-enterprise vehicle application frameworks.

6.1 MDE based Domain Specific Interface Metamodel Semantic Alignment Approach: An Overview

While models describe a specific abstraction of reality, metamodels are models of languages used to define models and can also be considered as object-oriented data structure in which models are stored [3][4]. The core of an MDE based EMF is an ecore. Since metamodels are also models metamodeling languages are needed, to describe modelling languages. Here the abstract syntax is described by a meta-metamodel. The EMF uses its own meta-metamodel to describe metamodels. As already mentioned in the earlier subsections, models, metamodels and meta-metamodels are arranged in a hierarchy of four layers (M0, M1, M2 and M3). Fig. 54 depicts a typical four-layered model hierarchy based on OMG standard.

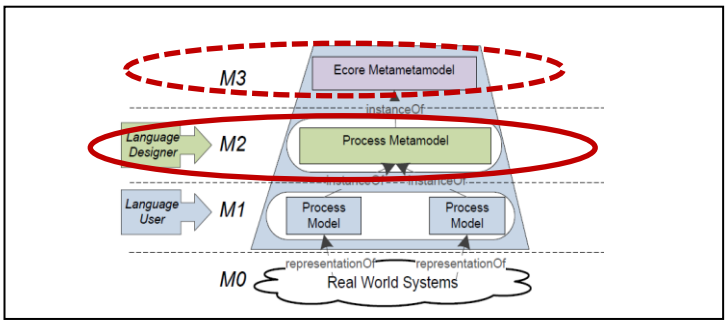


Fig. 54. Representation of the ecore Metametamodel (MOF) with standardized four layered modeling architecture.

6.1.1 Ecore: The Rationale

As described in earlier subsections (2.2.1), Ecore is an implementation of EMOF defined in the Eclipse Modeling Framework [22]. For example, while EMOF defines one class for defining properties, Ecore defines two types of structural features: attributes and references. The practical aspects inherent in Ecore make it more suitable for adoption. Fig. 55 illustrates the fundamental classes of a basic Ecore model.

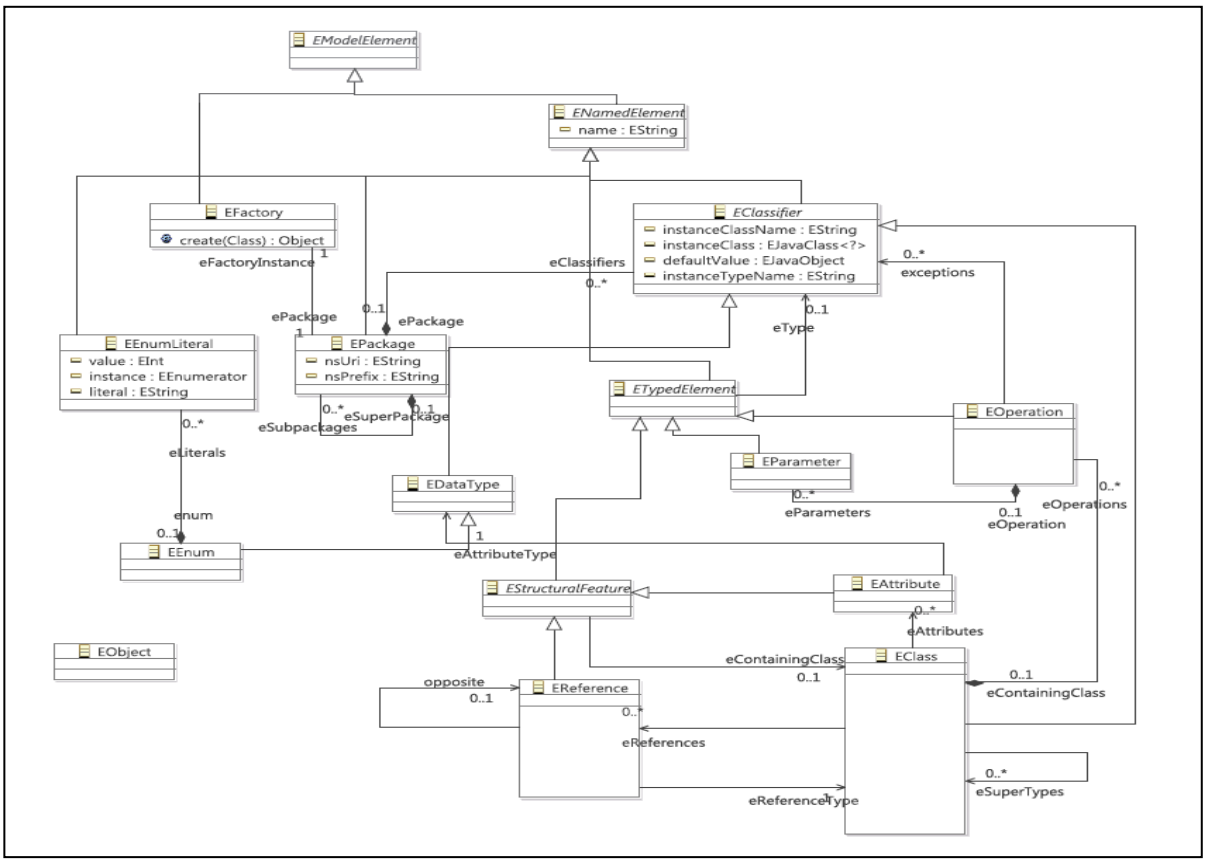


Fig. 55. Overview of an ecore metamodel.

The class *EModelElement* allows to tag model elements with names. In *EPackage* is an EModelElement that contains classifiers and sub-packages. Properties are defined by references and attributes as structural features. An *EReference* is a type of structural feature that has as type an *EClass*. An *EAttribute* is a type of structural reference that has as type an *EDataType*.

In EMF, Ecore can exist on both the M3 and M2 level of the metamodel hierarchy, as it allows for the definition of metamodels, but may also be used to define models at the M1 level. This is different from MOF, which may only be used to define metamodels. EMF comes with capabilities to serialize and deserialize models defined in Ecore to and from XMI [5]. Besides Ecore, there are several tools and frameworks developed on top of EMF that enable model-to-model and model-to-text transformations.

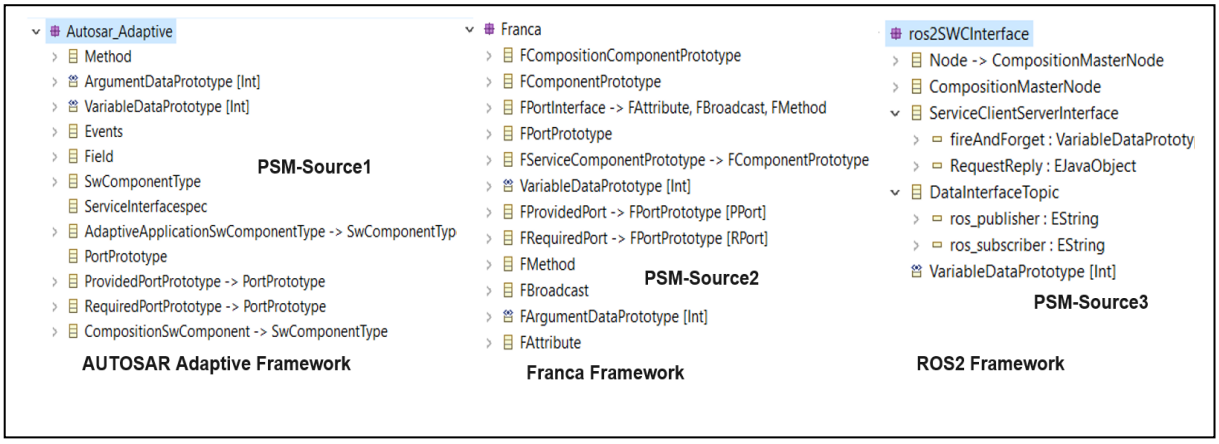
6.1.2 Semantic Mapping of Component Framework Interface Ecore Metamodels

It is relatively easy to find semantic correspondences between ecore as M2 level metamodel and Ontology Definition Metamodel (ODM) such as OWL2 (OWL version 2) metamodel, as both formalisms are fit for conceptual modeling [5]. Both ecore and OWL addresses conceptual metamodels where semantically related concepts are to be identified between models of a domain. Despite of possible similarities in semantic interface traits between the heterogeneous automotive domains component framework source metamodels, MDE approaches such as *EcoretoEcoreMapping* cannot semantically map or explore the semantic interface traits similarities between the platform-specific source ecore metamodels due to their structural heterogeneities.

Demonstration of Ecore to Ecore Metamodel Semantic Mapping using a Case Study

In context of semantic mapping of platform-specific software component interface metamodels for interoperability using MDE based approach, three most common automotive heterogeneous knowledge domain component framework interface Ecore metamodels are considered as three sources for the illustration of the case study. Mostly used vehicle application component frameworks for the given case study are namely, AUTOSAR Adaptive from automotive knowledge domain, Franca from infotainment knowledge domain and ROS2 from robotics knowledge domain. Hence, for efficient collaborations between these application SWC frameworks, we selected the SWCs' interface models of these three platform-specific frameworks for semantic mapping and named them as Source 1, Source 2 and Source 3. AUTOSAR Adaptive application framework software component interface Ecore metamodel has been considered as a PSM-Source1, Franca (including Franca+) infotainment application framework software component interface ecore metamodel has been considered as a PSM-Source2 Ecore metamodel and Robotics application framework ROS2.0 software component interface Ecore metamodel has been considered as a PSM-Source3. Fig. 56 depicts the *Ecore* interface metamodels of the three considered sources.

Fig. 56. Illustration of case study's platform-specific vehicle application SWC frameworks' interface ecore metamodels.



Despite of semantic equivalence between the platform-specific interface models, however, due to the heterogeneity in interface PSM API structural architectures and in semantic notations, a direct semantic mapping, say, PSM A to PSM B is not possible. Using a domain specific, intermediate platform-agnostic interface metamodel DM as a shared PIM between the semantically compared interface PSMs is essential for ecore to ecore metamodel mappings of PSMs. As seen in Fig. 57, DM contains generic domain interface semantic traits that are common to the semantically compared interface PSMs resources considered for case study [8].

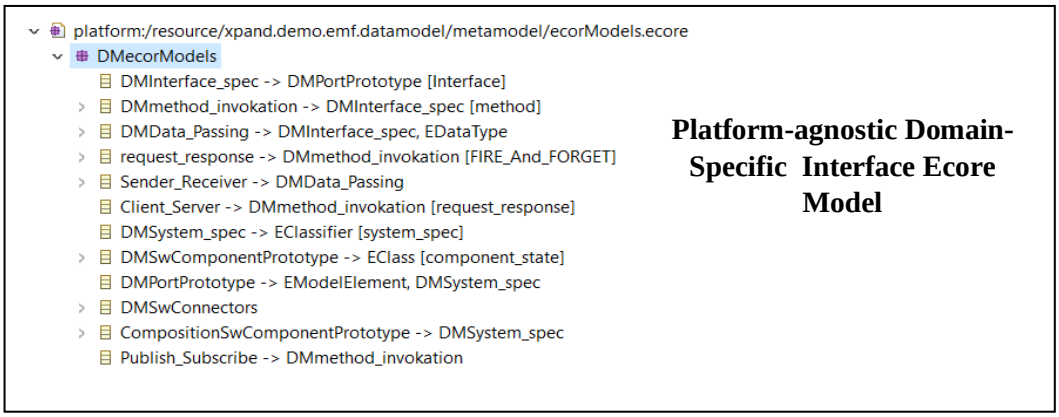


Fig. 57. Abstract representation of platform-agnostic, vehicle domain-specific generic SWC interface ecore metamodel.

However, in context of current scope of SWCs semantic interoperability, the uniqueness of each PSM SWCs' interface is essential to be considered, therefore, direct model to model transformation, say, from PSM A to PSM B using MDE based model transformation tools is not considered in the current scope.

The platform-specific heterogeneous knowledge domains source metamodels, that is, Source 1, Source 2 and Source 3 are initially manually mapped to the intermediate Platform-independent Model (PIM) domain reference interface metamodel agent, DM, as seen in Fig. 58.

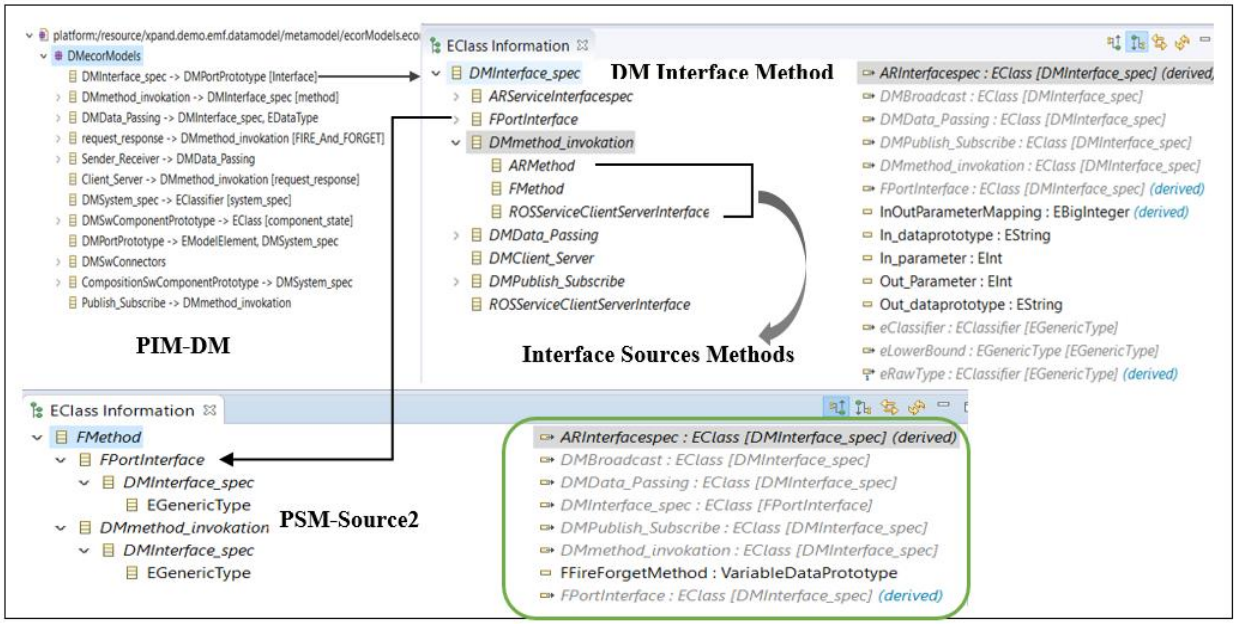


Fig. 58. Abstract representation of semantic alignment of interface sources using MDE based ecore to ecore mapping approach.

The intermediate reference model represents directly semantic data that are common to the source metamodels and is used to handle the semantic heterogeneities between them. The DM interface PIM contains generic domain interface traits that are semantically common to interface traits of PSM-Source1, PSM-Source2 and PSM-Source 3. Due to this semantic commonality, the platform-specific interface traits classes of metamodels of Source 1, Source 2 and Source 3 are declared as child class members of the polymorphic interface traits classes of DS metamodel by using the ecore metamodel property *ESuperTypes*, for example, *ARMethod* of PSM-Source1, *FMethod* of PSM-Source 2 and *isService* of PSM-Source 3 are class members of DM polymorphic class *DMmethod_invokation* due to semantic commonality, also can be seen in Fig. 59 [11].

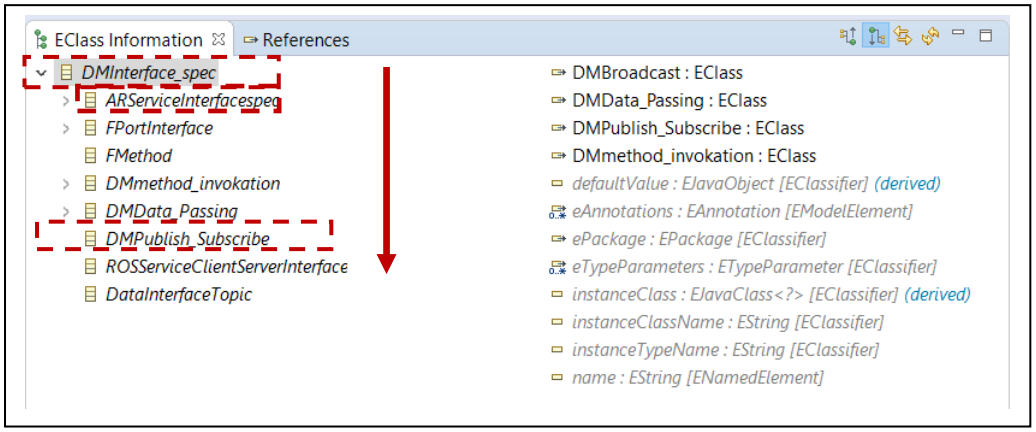


Fig. 59. Reference domain-specific interface ecore metamodel (DM) representing polymorphic interface semantic traits.

However, due to the fact that implicit knowledge cannot be explored owing to the lack of reasoning or query mechanism, the semantic synergies between the interface traits of PSM-Source1, PSM-Source2 and PSM-Source3 cannot be exploited completely.

That is, for example PSM-Source1 *ARServiceinterfacespec* can be semantically mapped to *DMInterface_spec*, similarly, PSM-Source2 *FPortInterface* can be semantically mapped to *DMInterface_spec* and PSM-Source3 *ServiceClientServerInterface* and *DataInterface* can be semantically mapped to *DMmethod_invokation* and *DMPublish_Subscribe of DMInterface_spec* manually. However, apart from inheritance or *derived* relation, no *inferred* semantic relations between the instances of semantically similar or equivalent interface metamodels method classes of PSM-Source1, PSM-Source2 and PSM-Source3 are possible to be explicitly expressed, despite of semantic similarity as also depicted by Fig. 60.

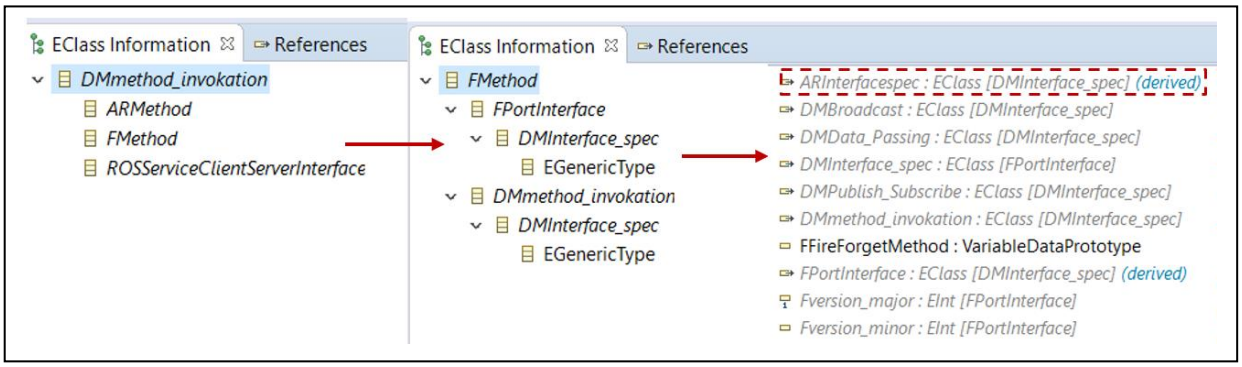


Fig. 60. Representation of SWC frameworks' interface method classes and their derived semantic associated relationships.

6.1.3 Challenges of MDE based Semantic Mapping Approach for Interface Ecore Metamodels

Domain analysis addresses the analysis and modeling of variability and commonalities of systems or concepts within a domain. Taking this into account, in perspective of improvement of reusability and interoperability between vehicle domain component frameworks interface models, it is absolute essential to focus on the domain components interface models independent of platform-specific API modelling languages and focus on checking the consistency among the terminologies or concepts of domain of interest [4][2].

Although MDE (Model Driven Engineering) claims that by using Platform Independent Models (PIMs) that abstract away the platform-specific details, the integration and interoperability across various platforms should become easier to produce in a platform-agnostic way, nevertheless, the equivalence mappings between the domain platform specific components interface models still cannot be explicitly expressed using semantic relations like generalization, polymorphism, etc. between the defined model classes [2][1].

In the past few years, in computer science, most MDE based modeling methods that are designed for model-driven systems using the Eclipse Modeling Framework (EMF) often lack proper visualization and focus instead on the

capabilities of model transformations and code generations. Ecore allows for relating classes by specialization relationships using metamodel properties like relational attributes or *EReferences*. However, when two classes of two ecore interface metamodels, say, C_A and C_B are semantically similar from functional perspective and are child class members of the same generic, polymorphic interface, metamodel third class, say, C_C in this situation, there is no inferred semantic relation possible to be expressed between the instances of the semantically related classes C_A and C_B. As an additional finding, ecore does not provide constructs to relate the sets of links described by a reference.

The strictness and consequence of inflexibility of metamodels using MDE based modelling tools makes it challenging to explicitly exploit polymorphism mechanism between the semantically equivalent interface traits of platform-specific interface metamodels. Due to the absence of derived concept hierarchies from logically defined metamodel class axioms, semantic relations like generalization or equivalence cannot be explicitly expressed between the semantically compared interface PSMs, as illustrated in Fig. Moreover, using a model driven approach to tackle domain specific semantic interoperability, MDE has no means (or no modeling tools) to formally describe domain concepts in a extensible way so that reasoning can be done about them and no means of checking the consistence and discovering conflicts among terminologies of domains of interest.

Nonetheless, ontologies can help to deal with the semantic alignment task on a solely conceptual level. Moreover, by defining domain concepts as first-class citizens, ontology allows developers to reuse the domain concepts on different situations independent of platform-specific details, thereby providing improvement in productivity and interoperability [10]. Understanding the strengths of ontology technologies in comparison to MDE like domain conceptualization using shared vocabulary, automated reasoning, inferred axioms, dynamic classification and consistency checking are essential for leveraging the development of promising solutions to semantic interoperability [1][3].

6.2 Possible Solution to Challenges of MDE based Semantic Mapping Approach: Extension of Interface Metamodels to Ontologies for Semantic Alignment

In general, MDE consists of the following two main artifacts: Modelling languages, which are used to describe a set of models and model transformations, which are used to translate models represented in one language into models represented in another language. In 2006, Berners-Lee et al. [32] observed an increased need for data integration, shared semantics and a web of data. They discussed how multiple heterogeneous datasets from various heterogeneous sources must be integrated. Ontologies are commonly used to integrate datasets. For instance, OWL is extensively used in the life sciences community for ontology development and data interchange.

Domain conceptualization using domain vocabulary and logical definitions is a key aspect of ontology technology. It is insufficient, just to have a robust physical infrastructure for transmitting data between vehicle applications, as the very same data can mean quite different things in different application frameworks depending on the platform. Hence, there is a necessity to use domain vocabulary to semantically map component interface models with one another. Like MDE technology, ontology uses OWL conceptual metamodels to exploit the semantic relations between the models of a domain. The OWL metamodel which is also part of the ODM specification as defined in the OMG standard, is implemented by extending the RDF (Resource Description Framework) and RDF (RDF Schema) metamodel. RDFS serves as a meta-language that defines itself and OWL metamodels at M2 level of four-layered modelling architecture [1][12]. Semantic Web languages, such as RDF and OWL facilitate interoperability in significant ways and provide formal mechanisms to express logical equivalences between classes and properties of the metamodel represented as ontology.

6.3 Strengths using Ontology based Approach: In contrast to MDE

Modeling issues like formal semantics or interoperability motivated the development of ontology languages. Formal semantics constrain the meaning of models, such that their interpretations are the same for different persons (or machines). Description Logics underpin the W3C standard Web Ontology Language (OWL) and provide the foundations for ontology languages. OWL together with automated reasoning technologies provides a powerful solution for formally describing domain concepts [66] .

In contrast to MDE, ontologies are developed to be manipulated by inference engines. OWL is capable of explicitly inferring dynamic generalization and specialization between interface traits classes as well as class membership of object based on the constraints imposed on the properties of class definitions [2][3]. The inference (or inferred axioms) and seamless automated reasoning are capable to explore semantic mappings among multiple ontologies.

Most of the OWL reasoning such as *asserted* or *inferred* axioms, etc. can be verified using W3C standard SPARQL query engine [66].

Role of reasoning for software modeling

Since in the context of MDE ontology technologies are used in software modelling it has become evident that there is a significant demand for software modelling environments which provide more sophisticated explanation services. In particular, the generation of explanations, or justifications, for inferences computed by a reasoner is now recognized as highly desirable functionality for both ontology development and software modelling. A language user designer developing (meta-) model recognizes an entailment and wants to get an explanation for the entailment in order to get the reason why the entailment holds (understanding entailments). If the entailment leads to some inconsistency or unsatisfiable classes, the user wants to get some debugging relevant facts to some inconsistency or unsatisfiable classes, the user wants to get some debugging relevant facts and the information how to repair the ontology. Hence, reasoning is vital for the formal description off the models.

Summary

Comparing the Ecore language and the OWL2 language, (1) An Ecore metamodel allows for specifying classes, datatypes and relations like references or attributes same as OWL2 metamodel specification language. 2) Classes in Ecore correspond to classes in OWL.With respect to the intensional semantics both class concepts represent a set of instances. (3) The references in Ecore correspond to object properties in OWL. They both represent sets of relations between instances of given types. (4) The attributes in Ecore correspond to data properties in OWL since they both describe relations between instances and values of a predefined datatype. (5) The datatypes of attributes in Ecore semantically comprise atomic values like the datatypes in OWL. OWL 2 ontologies additionally allow for modeling of individuals and assertions. (6) With respect to the intentional semantics instance in a domain model correspond to individuals in an OWL2 ontology. Both are classified by classes. (7) In Ecore model links are instances of references and represent connections between two instances. In OWL 2 object property assertions are used to define connections between two individuals with respect to an object property. (8) In domain models attribute assignments define the relation between an instance and a value. In OWL 2 data property assertions are used to define the relation between an individual and a value with respect to a data property.

However, to simplify the design approach using the proposed methodology, for the ontology language OWL2, only those constructs, which are replaceable by respective counterparts in Ecore-based metamodels are considered. In the case of the Ecore metamodeling language only those constructs, which are directly representable in an ontology are mentioned.

Chapter 7 Extension of MDE based Modeling Approach to Ontologies for Evolution of Domain Unified Interface Ontology

For the implementation of the possible solution design approach in Chapter 6, in this chapter, an semi-automated approach is proposed to extend vehicle application domain heterogeneous SWC framework interface metamodels to ontologies using a transformation bridge in order to carry over the advantages from ontology technologies such as explicit semantic relations to MDE based software modeling domain. Using a transformation bridge, the basic constructs of component frameworks interface metamodels described by Ecore are mapped to corresponding constructs of OWL metamodels.

7.1.1 Overview of Methodology for Metamodel Transformations between Ecore and OWL

The proposed methodology bridges the gap between pure modelling language metamodels [1] and OWL metamodels in order to reuse the knowledge from both MDE and ontology technology-based paradigms. For the semantic alignment of automotive domain component framework interface models for interoperability, the proposed methodology follows different steps, as illustrated in Fig. 61.

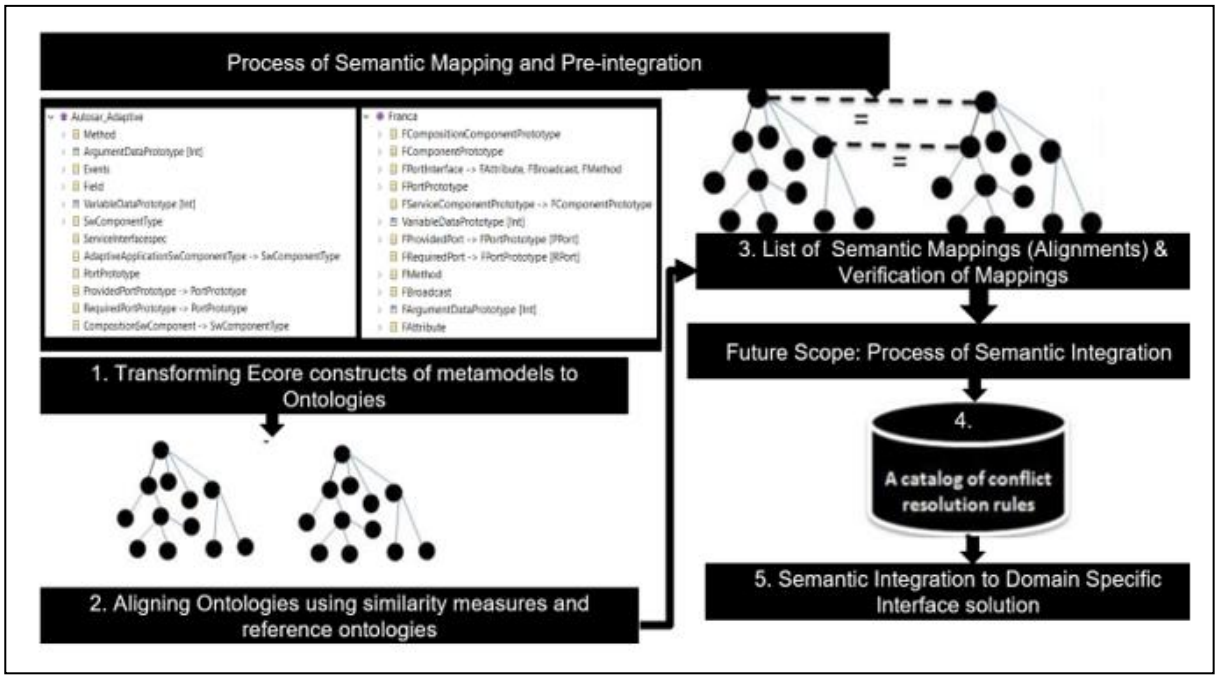


Fig. 61. Overview of methodology concept for semantically ontology mapping and integration approach.

The proposed methodology bridges the gap between pure modelling language metamodels [1] and OWL metamodels in order to reuse the knowledge from both MDE and ontology technology-based paradigms. For the semantic alignment of automotive domain component framework interface models for interoperability, the proposed methodology follows different steps, as illustrated in Fig. 61. In the first step, an approach to mapping of constructs of interface metamodels described by Ecore to OWL ontologies was proposed [2][11]. As a second step, the given approach was extended to explore the possible semantic alignments between the resultant OWL interface ontologies using OWL reasoning and inference engines. In the last step of the proposed methodology, and further we propose to verify and validate the possible inferred semantic synergies between the OWL interface ontologies using the same case study from Chapter 6 and SPARQL queries. Fig. 62 illustrates a generic comparison between ecore and OWL2 metamodels specifications based on OMG standard four-layered modeling architecture.

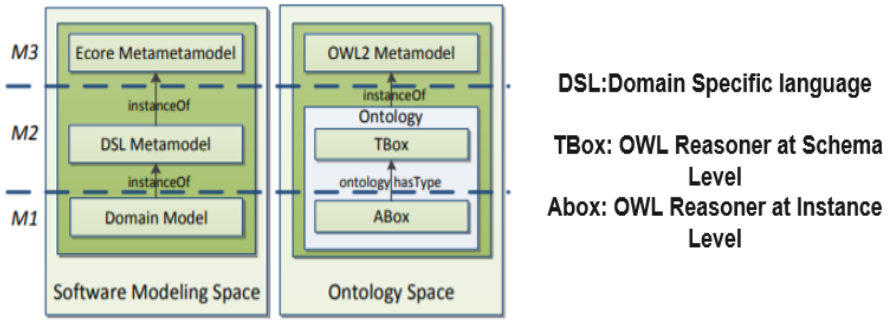


Fig. 62. Generic comparison between ecore and OWL2 conceptual metamodeling specification layers .

7.1.2 M3 Transformation Bridge: Mapping Component Framework Interface Metamodels constructs from Ecore to OWL

The migration of metamodeling languages constructs or abstract syntax like classes and properties described by ecore to corresponding OWL constructs can be achieved by implementing a M3 layer transformation bridge [11][10][7] between the software languages as can be seen in Fig. 63 [62]. The knowledge base for the implementation of a M3 transformation bridge [3] for automotive component framework interface metamodels can be summarized by the TABLE XIII.

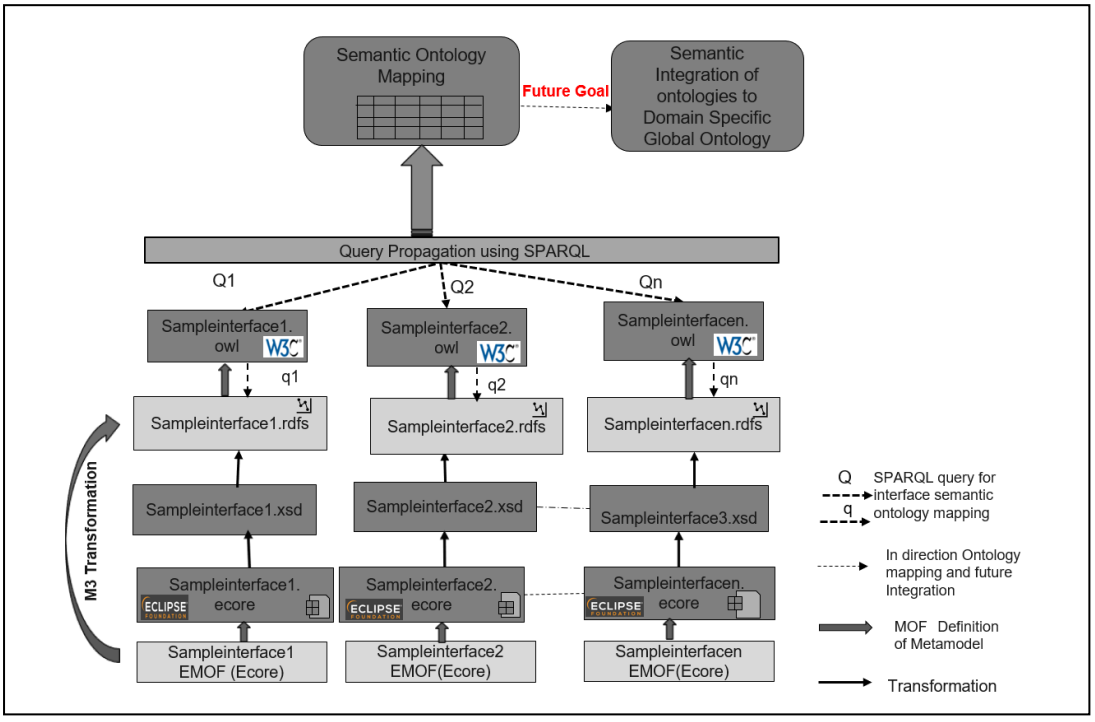


Fig. 63. Abstract layered representation of M3 Bridge approach to transform interface metamodel constructs from ecore to OWL2.

At the application component level, data interoperability relies on the semantic alignment or mapping between the various component framework interfaces data models represented as XML schemas (XSD). With the XML schemas being the preferred standard for the interface description exchange between most of the automotive application domain components, however, the data interoperability between the semantically equivalent but structurally different data constructs of multiple heterogeneous XSDs stands as a challenge in the absence of an ontology-based approach. An XSD (XML Schema) can be seen as a metamodel for XML documents, considering this, the EMF is cable of generating XSDs based on Ecore metamodel specifications . To simplify the one-to-one mapping of metamodel constructs from Ecore to OWL2 and to prevent loss of information, schematic translation of, constructs from XSD to ontology based RDFS is considered in the scope of the proposed approach as illustrated in TABLE XIII. .

TABLE XIII. SUMMARIZED MAPPING TABLE FOR SEMANTIC MAPPING OF META-MODEL ENTITIES FROM ECORE TO OWL2

Ecore	XSD	RDFS	OWL
EObject →	xsd:namespace	rdf:resource →	OWLNamedIndividual
EClass →	xsd:element type	rdfs:class rdf:property →	OWLClass
ESuperTypes →	xsd:extension	rdfs:subClassOf →	RDFSSubClassOf
EAttribute →	xsd:attribute name type	rdf:property rdf:about rdf:range →	OWLDatatypeProperty OWLDataRange
EReference →		→	OWLObjectProperty
EEnumLiteral →	xsd:enumeration value	rdfs:label →	RDFSLiteral, OWLLiteral
EDatatype →	xsd:datatype	rdf:datatype →	RDFSDatatype
EAnnotation →	xsd:annotation	rdfs:label →	RDFSLabel

7.2 Implementation of M3 Transformation Bridge

The implementation of an M3 transformation bridge consists mainly of identifying concepts in the vehicle SWC frameworks' interface ecore metamodel and the OWL metamodel which can be mapped to each other.

7.2.1 Step1: Translation of Ecore to XML Schema (XSD)

With the XML schema or XSD being the preferred standard for the interface information exchange between most of the vehicle applications' domains software components, it is meaningful to translate the interface ecore metamodels to XML schemas respectively using the ecore EMF generator modeling tools associated with MDE based Eclipse IDE platform, as shown in Fig. 64. Also from a logical perspective, in the scope of metamodel transformation from ecore to OWL2, schematic translations such as ecore to XSD and XSD to RDFS must be considered to prevent enormous loss of information as the existing XML Schema datatypes may be used in OWL ontologies if they have been already declared in them.

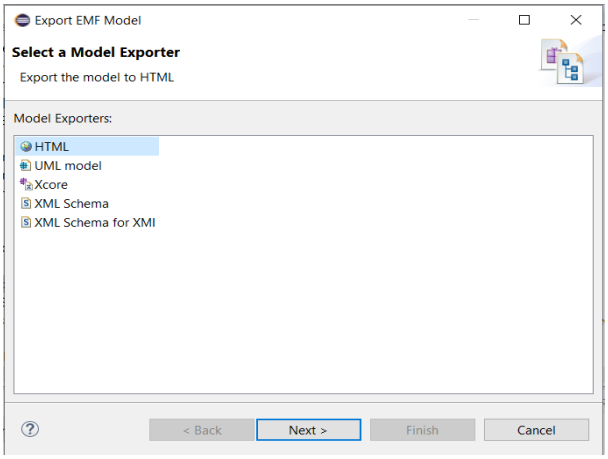


Fig. 64. Overview of exporting ecore model to XML Schema (XSD) using emf tools.

Despite of representation of same information, XSD represents abstract meta data compared to ecore elements. This is because XML Schema naming conventions are less restrictive then Javas' (and consequently ecores'), names sometimes need to be converted to conform to the naming conventions outlined in the XML standard specification. Fig. 65 illustrates an example showing mapping between an ecore *package* to XML schema using emf tooling support.

```
<xsd:schema
   targetNamespace="http://www.example.com/libary">
   …
</xsd:schema>
```

```
EPackage
   name="library"
   nsPrefix="library"
   nsURI="http://www.example.com/library"
   …
```

Fig. 65. Equivalent representation of an EPackage with XML schema.

A typical example of exporting and mapping a vehicle application SWC framework's interface ecore metamodel constructs to equivalent XSD constructs can be seen in Fig. 66.

Fig. 66. An example on exporting ecore metamodel constructs to equivalent XML schema using emf tools.

Since the XSDs are automatically generated from the interface ecore metamodels, therefore, the data interoperability between the semantically equivalent but structurally different data constructs of multiple heterogeneous XSDs still stands as a challenge in the absence of advanced semantic support like domain ontologies and logic-based reasoners.

7.2.2 Step2: Schematic Translation from XSD to RDFS

To confront the issue of data interoperability by exploring the possibilities of semantic alignments between the interface descriptions of various component framework templates, schematic translation of the XML schemas representing the various component interface ecore metamodels to RDFS (Resource Description Framework Schema) was considered to achieve advanced semantic support from domain conceptualizations and logic-based

reasoners [63]. Schematic translation of XSDs to RDFS bridge the gap between XSDs and OWL interface metamodel specifications by lifting the syntactic level of XML documents to the semantic level of OWL ontologies using ontology framework supported tools [65], as seen in Fig. 67. RDFS are ontology-based schemas that can be represented as an object model or a kind of constrained relational model and can provide semantic enrichment to XML schema data constructs, as seen in Fig. 68 [59][62].

Incorporating the XML and RDF paradigms approach was also proposed by authors et.al [60] but this approach did not consider any heterogeneous sources with different syntax or data models. The author developed an integrated model for XML and RDF by integrating the semantics and inferencing rules of RDF into XML, so that XML querying can benefit from their RDF reasoner. In context of domain specific global ontology, the author of [61] has proposed an ontology for automobile industry named as VCO (Vehicle Corporate Ontology) that would address the problems of platform and syntactic heterogeneity by mapping between individual schemas and XSLT transformations. The authors of [62] proposes an ontology-based framework for interoperating of two XML documents at semantic level and proposes integration of local RDF ontologies to a hypothetical global ontology. A fully automated mapping of ontology into a relational Database schema with a complete mapping approach was also proposed by authors of [64]. The intention to map and translate the XSD nested structure into the relation-based structure such as RDFS expressed by ontologies is to enable semantic enrichment and semantic alignment between the semantically equivalent data constructs conforming to different XSDs by hiding their structural heterogeneity in the native nested structure and exploring areas of possible semantic synergies [60].

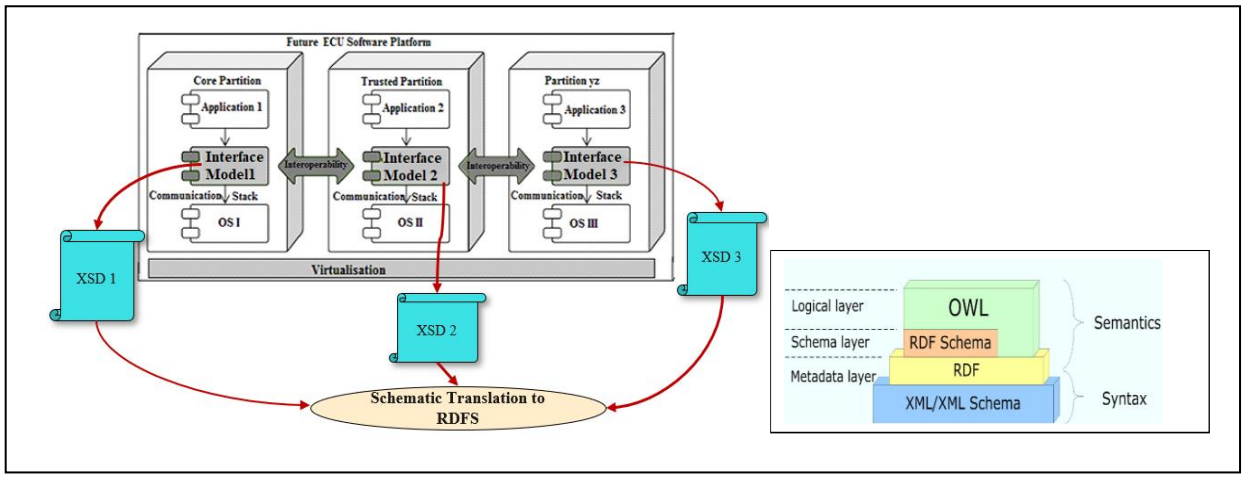


Fig. 67. Overview of schematic relation between XSD and RDF Schema (RDFS) .

```
<rdf:Description rdf:about="#r-14-0-0">
    <composite:index
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</composite:index>
    <sxsd:base>Franca:FComponentPrototype</sxsd:base>
    <rdf:type rdf:resource="http://topbraid.org/sxsd#Extension"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.topbraid.org/2007/05/composite.owl#index">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:comment>The index of the subject among its siblings. Note that this could in
principle take arbitrary numbers (including floats). We recommend using xsd:ints starting
at 0.</rdfs:comment>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="#r-17-0-0">
    <composite:index
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</composite:index>
    <sxsd:base>Franca:FPortPrototype</sxsd:base>
    <rdf:type rdf:resource="http://topbraid.org/sxsd#Extension"/>
  </rdf:Description>
  <rdf:Description rdf:about="#r-4">
    <composite:index
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">4</composite:index>
```

Fig. 68. Illustration of schematic translation of XSD to equivalent RDF Schema using an example.

As an extension to the Step1, the various existing platform-specific component framework XML schemas representing the interface ecore metamodels with the basic information required for peer-to-peer interoperability are then schematically translated to RDF-based local ontologies using an ontology-based tool, as seen in Fig. 62 [7].[7]

The tool [65] uses an automated internally mapping technology to map each entity or construct in the RDFS to the corresponding construct in the XSDs representing interface ecore metamodel, as also seen in TABLE XIV. The schematic translation from nested (XSDs) to relational databases (RDFS) considers the constraints and structure of the target schema [64] [59][7]. Some of the mappings between XSD and RDF Schema constructs are summarized in the TABLE XIV.

TABLE XIV. SEMANTIC MAPPING BETWEEN XSD AND RDF SCHEMA CONSTRUCTS

Input Schema: XSD Construct	Target Schema: RDF Schema Construct
Attribute	Property
Simple type	Property
Complex type	Class
Element	Class
Cardinality Qualifier	Cardinality qualifiers
Extension	subClass Of axiom
Sequence	Unnamed Class

After the process of translating XML Schemas into generated OWL ontologies, ontology engineers must collaborate with domain experts in order to add supplementary semantic information, not expressed in the underlying XML Schemas, to domain ontologies. According to OMG's four-layered modeling architecture, a MOF class at the M3 layer, is used to define M2 layer classes, similarly, the RDFS (M3) constructs at the schema layer in Fig. 69 are further extended manually to define OWL2 metamodel (M2) constructs at the logical layer using the ontology tool.

```
<!-- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FComponentPrototype -->
    <owl:Class
rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FComponentPrototype">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FCompositionComponentPrototype"/>
    </owl:Class>
<!-- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FCompositionComponentPrototype -->
    <owl:Class
rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FCompositionComponentPrototype">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#Fsystem_Template"/>
    </owl:Class>
<!-- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FMethod -->
    <owl:Class
rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FMethod">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
49#FPortInterface"/>
    </owl:Class>
```

Fig. 69. Overview of translation of RDF Schema to OWL2 metamodel constructs using ontology framework supported tools.

7.3 Platform-agnostic, Mediator Centric Approach towards Exploration of Semantic Synergies between Component Framework Interface Ontologies

In contrast to MDE, explicit reasoning and inference engines are the known vital features of the ontology technology that provides benefits of semantic alignment and semantic integration [3][5] [12]. Taking this fact into account, as a next step to *M3 Transformation Bridge* (in detail in Sub-section 9.1), it is time to explore the semantic synergies between the various vehicle component frameworks' interface metamodels specified as OWL ontologies using an automated reasoner tool of an ontology IDE framework and semantically integrated the aligned interface ontologies. Also, validate the reasoning using the SPARQL query engine. The semantic mapping and alignment of interface OWL ontologies proposed approach is demonstrated using the same vehicle domain case study from Chapter 6. Semantic Web languages, such as RDF and OWL facilitate interoperability in significant ways and provide formal mechanisms to express logical equivalences between classes and properties of the metamodel represented as ontology. In consideration to ontological metamodel layers and metamodel language layers, there are two different approaches. One is the use of modeling language across all ontological layers. The other is on usage of different modelling languages at different ontological layers. In context of current scope, the former approach is considered for semantic alignment of vehicle component interface metamodels.

7.3.1 Ontology-based Interface Semantic Alignment and Integration using a Platform-independent Reference Ontology: An Algorithmic Representation of Fundamental Steps

Semantic data integration is the process of using a conceptual representation of the data and of their relationships to eliminate possible heterogeneities. In this subsection, some fundamental definitions are introduced before the illustration of the methodology using case study in the next subsections. In this direction, the following definition in the current research scope were adopted and illustrated in Fig. 70[81].

- **Definition 1**: *Each interface ontology is defined as a 4-tuple IO=<C, S, I, A>, where C is a finite set of Interface concepts; S is a finite set of semantic relations; I is a set of instances (or individuals); A is a set of axioms, which is expressed in an appropriate logical language such as SWRL, etc.* [81]
- **Definition 2**: *Interface semantic data integration system is a 5-tuple ISI=<O_{GI}, O_{SI}, D, M_{GS}, M_{SD}>, where*
  - *O_{GI} represents the platform-agonistic, vehicle application domain-specific, global conceptual interface ontology schema, expressed in an ontology language L_G such as OWL2. GI provides the global view for vehicle domain application users.*
  - *O_{PI} represents the local platform-specific framework interface conceptual ontology schemas, expressed in same ontology language L_G as O_{GI}. PI defines the local platform-specific semantics of interface data resources.*
  - *R_O represents a platform-independent mediator or reference ontology to ease semantic interoperability or semantic alignment between platform-specific interface ontology schemas. R_O defines the vehicle domain specific, generic application SWC's communication interface semantics.*
- **Building Mappings M_{PR}:** *define mappings between O_{PI} and R_O, that is O_{PI} ϕ R_O, where ϕ is a set of mapping axioms between R_O and $O_{PI.}$ For example, mapping of interface concepts like MethodCalls() types between the intermediate agent and the platform-specific interface conceptual schemas, as seen in Fig. 70* [81].
- **Building Mappings M_{GS}:** *define mappings between O_{GI} and O_{PI}, that is O_{GI} ¥ O_{PI}, where ¥ set of mapping axioms between each O_{SI} and global schema $O_{GI,}$ as seen in Fig. 70* [81].
- **Building Queries Q_{GI}**: *queries can be processed in two directions: the data-integration direction, the query on the global ontology is rewritten into subqueries over multiple sources, and the peer-to-peer direction, the query on some local interface conceptual schema is propagated to other local interface conceptual schema sources connected through the global ontology* [63][62]. *Both Q_{GI} and its subqueries(at local platform-specific interface conceptual schema level are expressed in SPARQL at schema level (also known as TBox Reasoning) and at individual or instance level (also known as TBox Reasoning).*

In this context of semantic interoperability, ontology plays important roles which can be concluded as follows: Firstly, it provides formal description for specific domain. Domain concepts and knowledge structure are formally defined by schema O_{GI}. Secondly, it provides semantic interoperability. Ontology mapping M_{PR} domain-specific, reference interface semantic and local platform-specific interface semantic. Lastly, it provides reasoning and deducing ability. According to the relationships among concepts defined in interface ontologies, implicit

knowledge can be inferred from ontologies to satisfy the users with incremental information. Towards semantic integration of semantically aligned interface concepts of local conceptual schemas, the global schema O_{GI} must essentially include[62]:

- *Merging of classes*: where multiple conceptually equivalent classes are combined into one class.
- *Merging of properties*: where multiple conceptually equivalent properties of a class are combined into one property.
- *Merging relationships between classes*: where conceptually equivalent relationships from one class c1 to another class c2 are combined into one relationship.
- *Generalizing related classes into a more general superclass:* The superclass can be obtained by searching an existing knowledge domain using reasoning.

7.3.1.1 Domain Interface Ontology Building: Methodology for Semantic Integration

For domain-specific semantic integration methods without ontology, the solutions always have the following drawbacks. Firstly, it has no means of checking the consistence and discovering conflicts among domain terminologies. Secondly, although the equivalence mappings can be realized, inheritance mechanism can't be implemented. Finally, implicit knowledge cannot be discovered owing to lacking reasoning mechanism. Therefore, ontology is often viewed as a key component to realize semantic integration of data [9].

The proposed methodology uses OWL ontology to represent the global semantics of domain interface model and the local semantics of heterogeneous platform-specific framework component interface data sources respectively based on the algorithm explained in the earlier subsection. The architecture and key components for implementing the methodology are presented.

To further illustrate the proposed methodology, this research report uses the same case study from subsection 6.1.2 to show how to integrate vehicle application component heterogeneous data sources (represented as local platform-specific interface ontology resources) semantically into global domain-specific interface ontology conceptual schema.

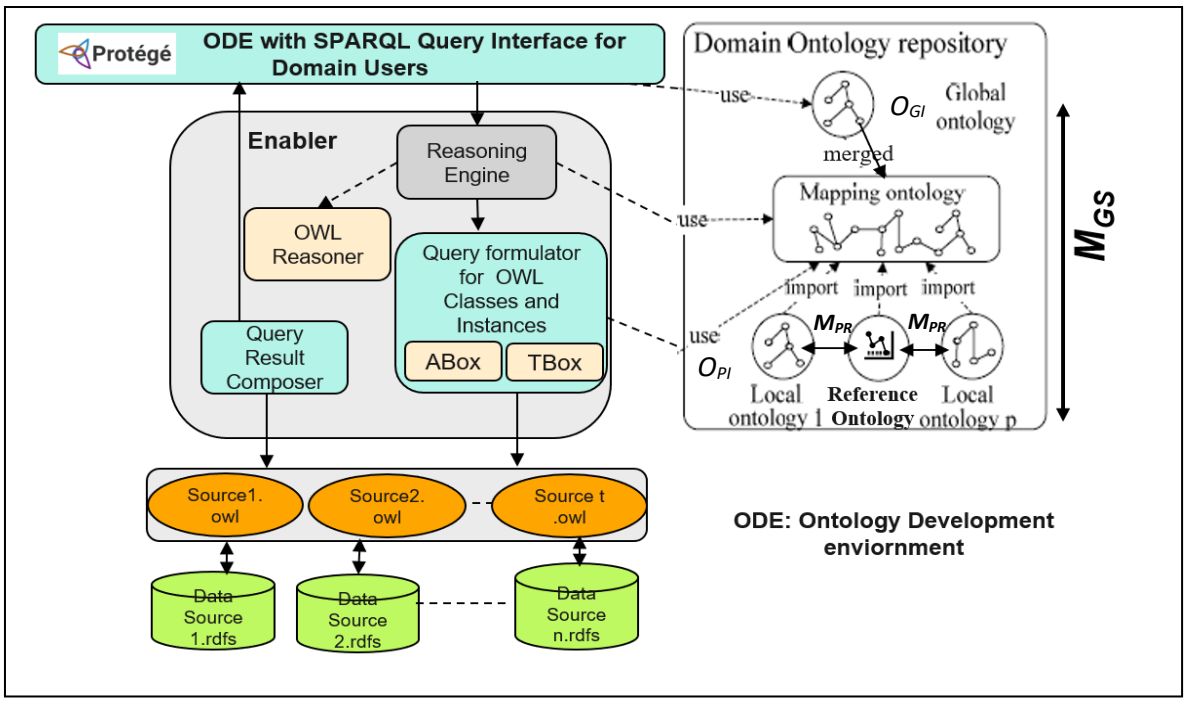


Fig. 70. Workflow model for semantic alignment and integration of SWCs' interface semantic ontologies.

To demonstrate the ontology-based interface metamodel semantic alignment approach, we use the same case study from subsection 6.1.2. The three semantically compared platform-specific interface models are represented as OWL2 metamodels or ontologies and named as ONT-Source 1, ONT-Source 2 and ONT-Source 3, as can be also seen in Fig. 71.

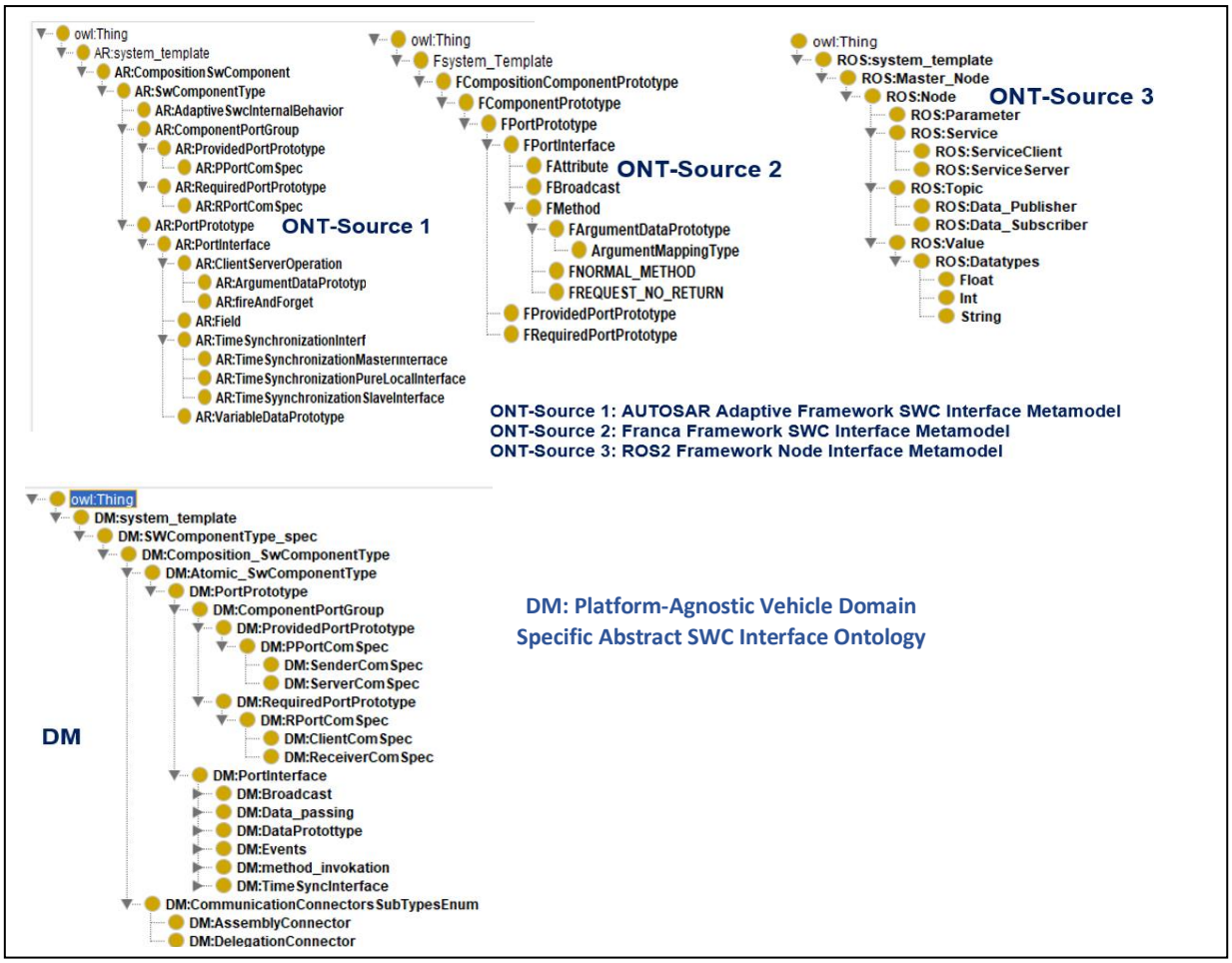


Fig. 71. Illustration of the case study SWC communication interface model using Thrift IDL.

As specified in the earlier subsection the fundamental definition of R_O, hence, to ease semantic interoperability, a shared, domain specific, platform-independent interface ontology, DM, is considered as an intermediate agent or reference ontology between semantically compared platform-specific interface ontology sources, as depicted in Fig. 72. The DM OWL2 ontology contains generic domain interface concepts that are semantically common to the interface ontology sources 1, 2 and 3[9].

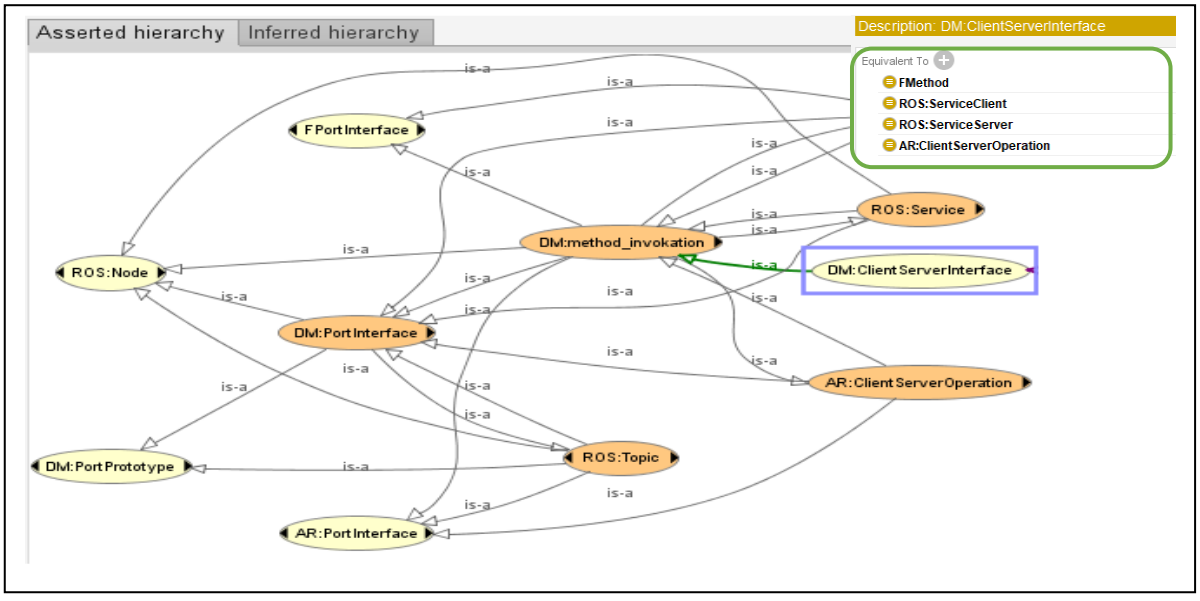


Fig. 72. Semantic mapping of SWC frameworks' interface ontological metamodels using asserted axioms .

Due to this semantic commonality, the entities representing interface concepts of local platform-specific conceptual schemas, namely ONT-Source1, ONT-Source 2 and ONT-Source 3 ontologies are asserted to be equivalent to the polymorphic interface traits entities of DM ontology, for example *ARMethod* type *AR: ClientServerOperation* of ONT-Source1, *FMethod* of ONT-Source 2 and *ROS:Service* of ONT-Source 3 are *asserted* manually to be semantically equivalent to *DM:method_invokation* [8] type *DM:ClientServerInterface*, as seen in Fig. 73 [42][43][9].

Merging of equivalent Classes, properties and Relationships using Semantic Integration of Platform-specific Interface Ontologies using Inferred axioms and Reasoning

Apart from the *asserted* object property axioms, the automatically generated *inferred* property axioms by the reasoner of an ontology framework semantically relates the various classes (also known as *TBox* Reasoning) and their instances (also known as *ABox* Reasoning) of ONT-Source 1, ONT-Source 2 and ONT-Source 3 using *equivalence* axiom, represented by the green arrows in the as seen in the Fig. 73.

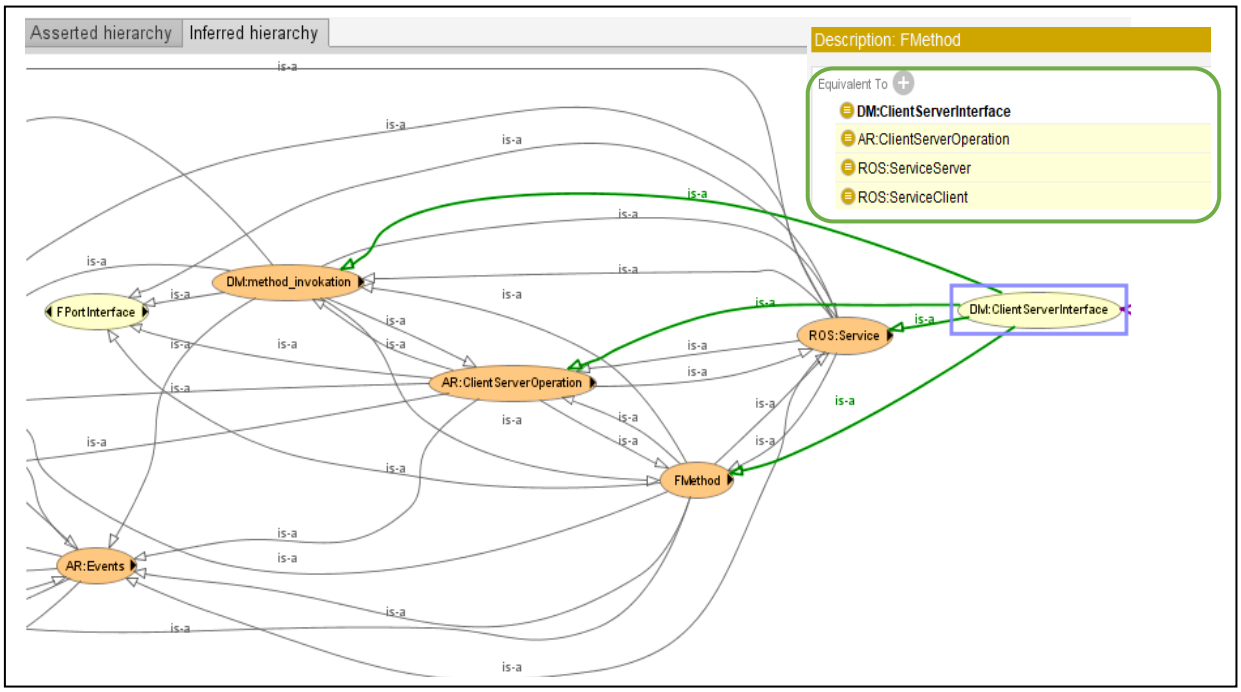


Fig. 73. Semantic alignment of SWC frameworks' interface ontologies using inferred axioms and reasoning.

That is, the inferred axioms automatically generated by the reasoner semantically aligns the Source 1 *ARMethod*, namely *ClientServerOperation* to Source 2 *FMethod* and to *ROS:Service* of Source 3, using *equivalence* (is-a) axiom feature of ontology-based modelling tools.

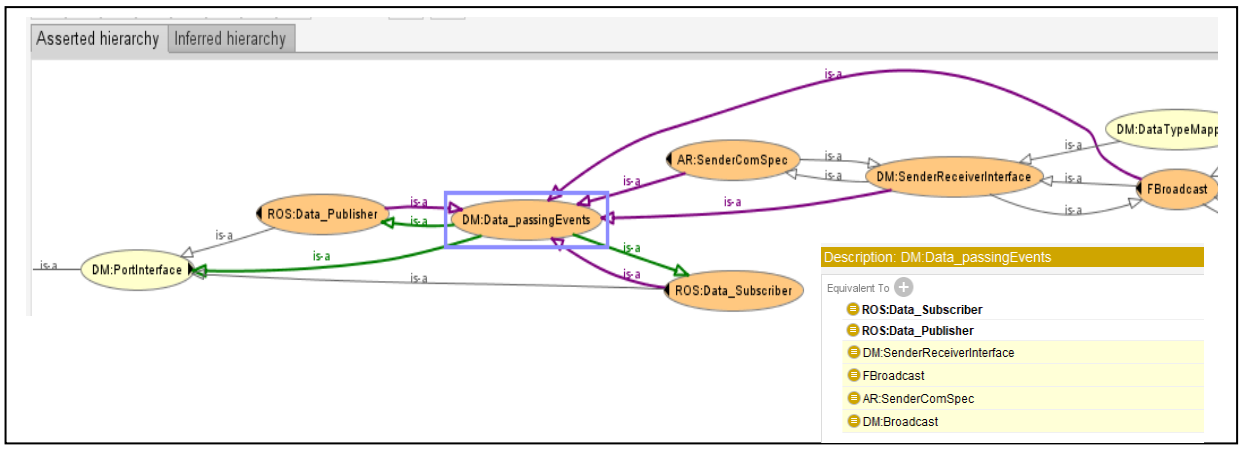


Fig. 74. Exploration of semantic equivalence relationship between data-passing interface method calls using inferred axioms.

An example of merging equivalent Classes, properties and Relationships using Semantic Integration of Platform-specific Interface Ontologies can be illustrated for *Data_Passing* interface *MethodCalls()* by the Fig. 74[9].

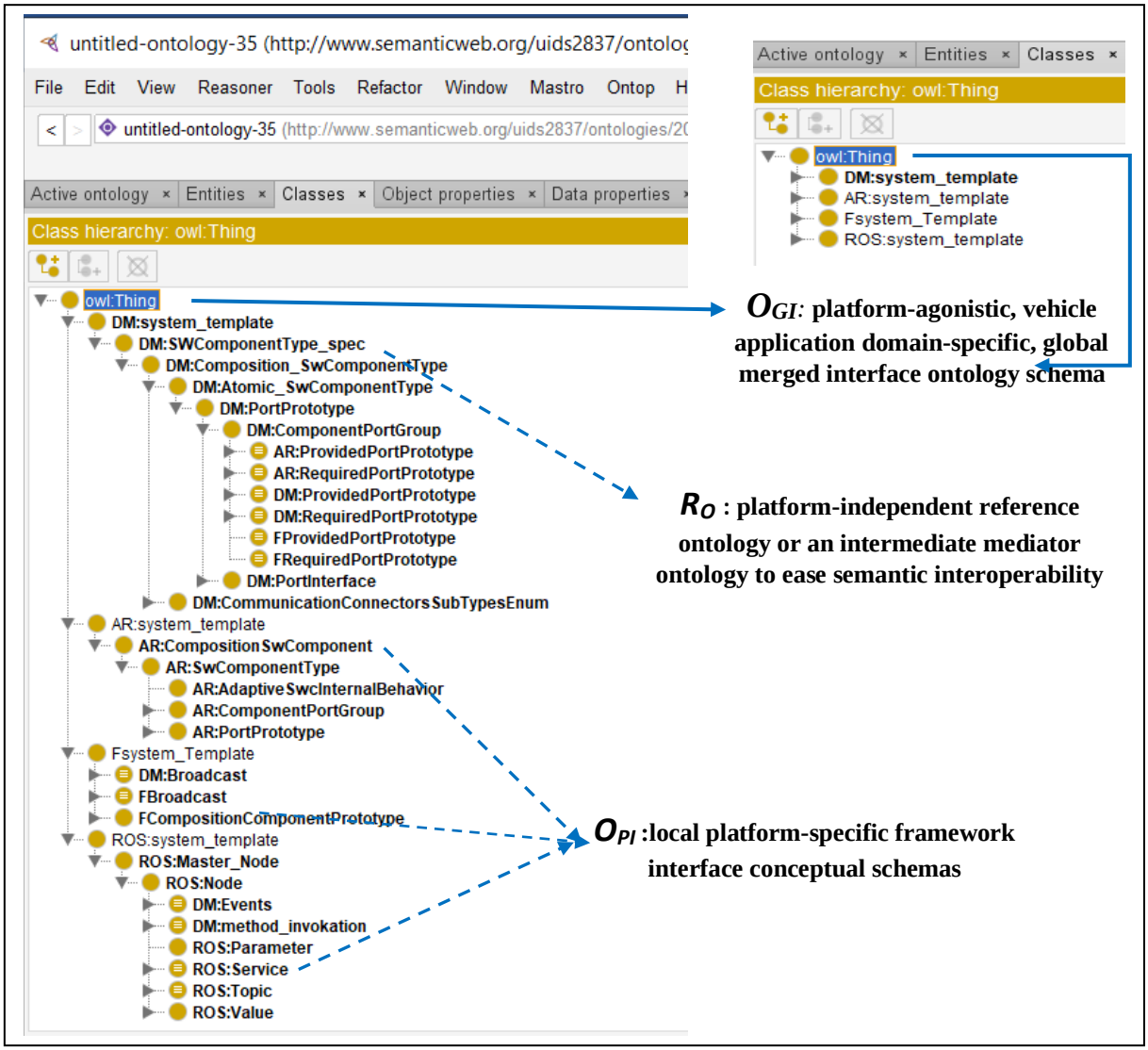


Fig. 75. Illustration of the conceptual global conceptual interface ontology schema for vehicle domain application SWC Frameworks.

In reference to Fig. 70 and the stated algorithm on semantic mapping and integration of ontology resources, the Fig. 75 illustrates the *Global Conceptual Interface Ontology Schema*, O_{GI} for the considered case study. All the *local interface ontology schema* resources, O_{PI}, are semantically mapped to each other using a platform and technology agnostic interface ontology mediator, R_O. After semantically mapped using semantic synergies in interface concepts, the local interface ontology schema resources are finally merged to the Global Conceptual Interface Ontology Schema.

7.4 Brief Overview of the Ontology Development Environment (ODE)

No matter what ontology representation metamodeling language is used, there is usually a graphical ontology editor to help the developer organize the overall conceptual structure and schema of the ontology such as adding concepts, properties, relations, and constraints; and, possibly, reconcile syntactic, logical, and semantic inconsistencies among the elements of the ontology. In addition to ontology editors, there are additionally other ontology tools as software plugins and number of reasoners available with the ODE in the ontology framework that help to manage different versions of ontologies, convert them into other formats and languages, map and link between ontologies from heterogeneous sources, compare them, reconcile and validate them, and merge them.

Protégé currently being the leading open source ontology development editor and environment and hence, considered in the current scope of research [19]. It facilitates the defining of concepts (classes) in an ontology,

properties, taxonomies, and various restrictions, as well as class instances, as illustrated in Fig. Furthermore, its uniform GUI (Graphical User Interface) has a tab for the creation of a knowledge acquisition tool for collecting knowledge into a knowledge base conforming to the ontology. Customizable forms determine how instance information is presented and entered. The knowledge base can then be used with a problem-solving method to perform various inference tasks.

Protégé supports several ontology representation languages, including OWL and RDF(S). Some forms of reasoning over ontologies developed with *Protégé* are also facilitated; for example, since OWL is based on description logics, inferences such as satisfiability and subsumption tests are automatically enabled [19]. Protégés' plug-in-based extensible architecture allows integration with a number of other tools, applications, knowledge bases, and storage formats such as storage back ends for UML (for storing Protégé knowledge bases in UML), XML Metadata Interchange (XMI) (for storing Protégé knowledge bases as XMI files), etc. as depicted in Fig. 76.

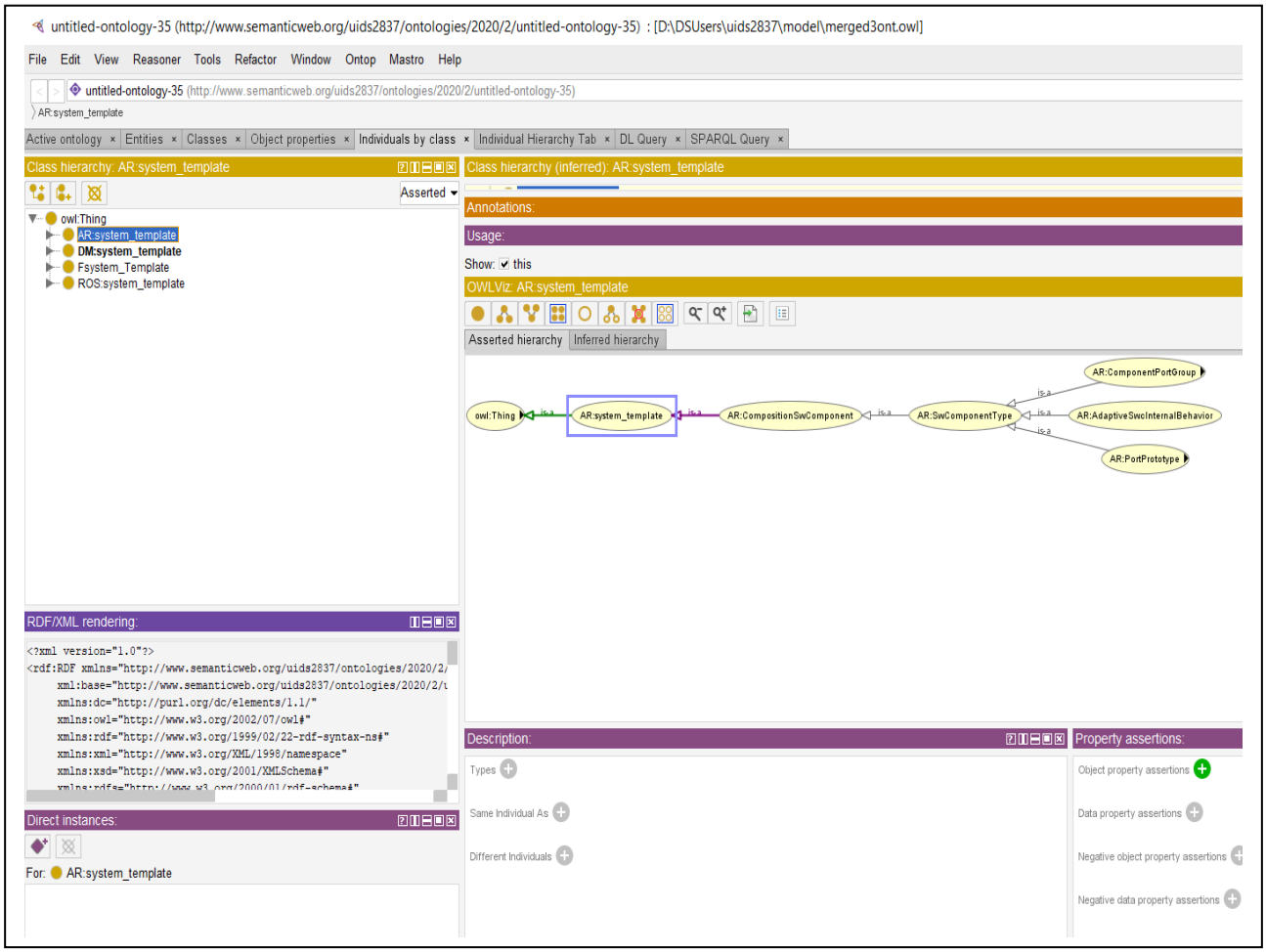


Fig. 76. Overview of Ontology Development Enviornment (ODE) using Protégé.

Summary

As illustrated in TABLE XV. , in contrast to OWL2 metamodels specifications, implied or inferred semantic axioms cannot be explicitly expressed with the ecore metamodeling method elements like classes, instances, attributes, etc. despite of the artifacts being hidden in the abstract metamodel and modeling languages [9].

TABLE XV. COMPARISON OF SEMANTIC ALIGNMENT QUALITY METRICS FOR MDE AND ONTOLOGY BASED CONCEPTUAL METAMODELING METHODS

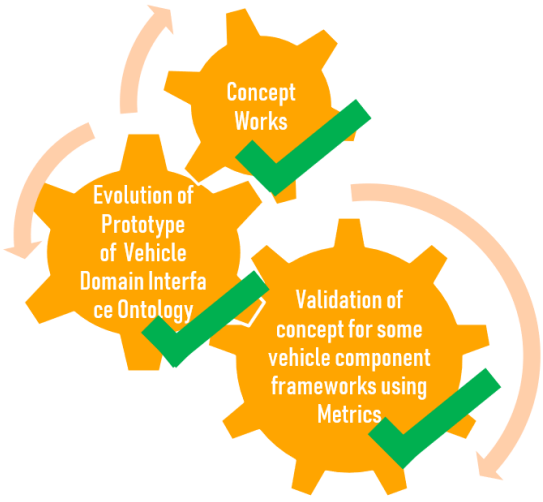
Quality Metric	Approach	Semantic Alignment Measurement	Ecore Metamodel	OWL2 Metamodel specification/Ontology	Performance Description
Semantic Association Navigability Metric (SAN)	Manual	Possible	√		**Possible** to identify the *derived* semantic relations between the **classes**.
Semantic Association Instance Metric (SAI)	Manual	Not Possible	√		**Not Possible** to identify the implied semantic relations between **instances** of classes.
Semantic Similarity Metric (SSM)	Manual	Possible		√	Implied/Inferred Semantic alignment relation between the **classes** at **schema level possible** to be measured.
Class Connectivity Richness Metric (CCR_i)	Manual	Possible		√	Implied/Inferred Semantic alignment relation between the **instances** of classes at **Knowledgebase level possible** to be measured.
Semantic Relationship Richness Metric (SRRI)	Manual	Possible		√	Semantic relationship between **instances** of semantically equivalent classes.

The above summarized table indicates that the ecore-based metamodeling method are mostly designed for narrow purposes which fits to the model-driven development domain but may not be suitable metamodeling method to tackle semantic interoperability when compared with ontology metamodeling methods. On the other hand, the usage of reference attributes is way more common in ecore-based metamodels. This fact could be explained by the purpose of ecore metamodels to act as structured data model. Hence, considering the above analysis on quality metrics, it can be concluded that both the metamodeling methods are complementary to one another.

Part IV Finale

For the semantic interoperability between service-based components' communication interfaces in vehicle domain, it is time to shift the focus from the implementation of a certain modeling language towards the explicit reification of the concepts covered by the languages [1]. From the viewpoint of cross-enterprise collaboration of services within the vehicle domain, it is also time to focus on the standardization of semantics ways of accessing services with various service models or profiles bounded with different platform-specific interface description languages, that means, the focus is towards standardization of the cross-enterprise services interoperability process in vehicle domain. To achieve standardization of vehicle services interoperability process, the significant research question that emerges is: ***What is the standard way to describe the interfaces of heterogeneous vehicle service components with varied service profiles?*** A possible solution to this research question is to evolve a domain ontology for representing a unified interface conceptual description for vehicle domain services from heterogeneous component frameworks. This helps to achieve better results in terms of semantic alignments and integrations as possible solutions to tackle interoperability issues.

Geared towards services semantic interoperability, in the semantic Web domain, OWL-S is an ontology within the OWL-based framework, which is used in conjunction with domain ontologies specified in OWL, provides standard means of specifying declaratively APIs (Application Program Interfaces) for Web services that enable automated Web service interoperability[35]. Also, in the Web domain, the Web Service Modeling Ontology (WSMO) is a conceptual model that provides an ontology-based framework which supports deployment and interoperability of services for semantic[35]. Due to the conceptual differences between typical Web service models (which are generally loosely coupled software applications) and vehicle service component models (which are generally tightly coupled software applications), implementing OWL-S process models or WSMO conceptual model perspectives, for service interoperability in vehicle domain, however, remains challenging and unanswered. The future extension of the work presented in this contribution would focus on finding solutions to address such challenges.



Chapter 8 Conclusion

Current automotive industry standards for describing vehicle Services focus on ensuring interoperability across diverse platforms, but do not provide a good foundation for standardization of service APIs definitions or descriptions. Over the last decade, in automotive domain, cross-enterprise interoperability is becoming a daunting impediment in context of providing efficient software IoT solutions at application level due to semantic data heterogeneity observed at application interface level. From a modeling perspective, the interoperability between heterogeneous service components' interface models within the same vehicle information system or ECU software platform would rely on semantic synergies between the components' interface models and resolution of possible semantic conflicts. MDE and ontology technology offers complementary solution to tackle semantic interoperability issues. Lack of semantic interoperability between vehicle service SWCs' interfaces causes major hurdles for precise service discovery, service invocation, service profile understanding, composing services, negotiating contracts and communications, etc. In the direction to ease semantic interoperability between vehicle service SWCs' interfaces, this work comprises of contributions of multiple facets towards generic, platform-agnostic, standardized semantic specification of vehicle service API models and also outlines the relevant state-of-the-art research so far conducted in this context of the current research scope.

Despite of few commonalities between ecore and OWL2 metamodel based semantic mapping approaches for vehicle domain service SWCs' interfaces, this contribution intends to highlight the comparison by depicting the differences between their semantic mapping approaches. It was revealed from the comparative static semantic analysis of MDE versus ontology based metamodel semantic mapping approaches that both the approaches are similar in the global process deployed to solve the semantic alignment and interoperability issues within a domain, however, the conceptual metamodeling styles of both approaches are quite different. This contribution focused on those aspects of the OWL2 based ontology metamodels which are not replaceable by respective counterparts in ecore based metamodels, for example, for a given SWC's interface metamodel, OWL2 allows for stating that two object properties are *equivalent* or *disjoint.* If two object properties are *equivalent* or *disjoint*, the sets of relations between their individuals they describe are also *equivalent* or *disjoint*. On the contrary, ecore based interface metamodels are not able to explicitly express such *equivalence* and *disjoint* semantic relations neither at class nor instance level using references or relational attributes. This is due to the fact, in comparison to ontology technology, the missing support of infinite reasoning capability and exploring inferred artifacts capability in MDE based semantic mapping approaches.

It can also be inferred from the comparative analysis and evaluation that the challenges using MDE based interface semantic alignment approach can be compensated by using the strengths of ontology-based interface semantic alignment approach. That is, to confront the crucial requirement for semantic interoperability and to leverage the development of an efficient solution, it is necessary to combine both MDE and ontology paradigms semantic alignment approaches in a value-added way by extending the MDE based components interface metamodels to ontologies. This means, the implicit concepts in MDE based vehicle service component API metamodels could be made explicit in ontologies, by utilizing automated reasoning, inference, and query engines of ontology technology.

With the above-mentioned perspective, this contribution proposes a semi-automated *Transformation Bridge* approach to extend the basic constructs of ecore based vehicle service components' API metamodels to corresponding constructs in OWL metamodels (M2 layered). Moreover, as vehicle domain complex and novel use cases involves manipulate heterogeneous overlapping domain knowledge frameworks like automotive, telematics, infotainment, robotics, cloud services, therefore, semantic alignment of cross-enterprise SOA frameworks' interface ontologies for interoperability using an *ontology mediator* centric approach represents a great interest in vehicle domain. The proposed design and implementation approaches are illustrated using appropriate vehicle domain case studies.

As a proof of concept to validate the results of interface semantic alignments and inferred axioms identified between the various vehicle service component frameworks' API ontology resources, the OWL reasoning capability was extended by using SPARQL query engine. It is understood that due to ever evolving heterogeneous descriptions of automotive SWCs artifacts that are frequently inconsistent, and tolerating this inconsistency is important if flexible collaborative working is to be supported for novel vehicle service requirements. Therefore, it is proposed to resolve any inconsistencies that are encountered with the change in vehicle service components'

artifacts during the semantic alignment implementation approach, at the time of detection. Nevertheless, still more work is required in directions to automate the proposed design and implementation approach in the future.

Exploration of semantic synergies between APIs of various vehicle applications SOA frameworks increases possibilities to reuse efficiently these various vehicle service components frameworks through their API semantic data in various types of semantic integrations within the vehicle domain in future. Although this research work is focused on vehicle domain, the method can give a good reference for heterogeneous data interoperability and semantic data integration in other science domains in the near future in perspective of design of novel complex IoT solutions.

Chapter 9 Future Work

In the recent years in automotive industry a plethora of wide variety of metamodeling techniques such as ecore, OWL2, etc. are being used. Perceiving this fact, it becomes significant to not only evaluate the quality of each of the metamodeling approaches but to also ensure validity of such approaches to increase the applicability or usage of these approaches in vehicle application domain. In principle, evaluation of semantic alignment quality for the SWC frameworks' interface metamodels (represented using Ecore and OWL2 metamodeling languages)) using metrics is substantial to guarantee that it meets the vehicle application domain requirements for cross-enterprise semantic interoperability. The evaluation of semantic alignment quality metrics ensures end-user to better understand whether a given ontology is suitable for his application domain. Therefore, as a part of the future work, the presented contribution would focus on the empirical study on the semantic alignment quality achieved using different metamodeling approaches to model SWC interfaces. This empirical study will be evaluated using semantic aware metrics. Based on the measured values of the metrics, several practical implications can be revealed that must be considered in perspective of future interface semantic integrations.

As it was presented in this research work, the ontologies are an especially useful formalism to specify vehicle domain SWC interface metamodels. Moreover, a direct translation to a grammar avoids manual and informal methodologies in the design phase of a new language. Getting inspired by WSDL and geared towards finding a unified representation for service-based APIs of vehicle application SWCs for interoperability, the work presented in this contribution will be extended in future in this direction. Therefore, the future work of the presented contribution will focus on the set of transformation rules to convert the ontological information into an AntLR grammar using Protégé and eclipse IDE or EMF supported tools, as seen in Fig. 77. Although, this perspective is not completely in line with the objectives of the current research work, however, in future, the work would be possibly extended to accomplish following objectives from applicability and usability enhancement perspective. The workflow for mapping of interface ontology to grammar and subsequent code will include:

- Attribute Grammar (AG) extraction from ontologies w.r.t ontologies' *Concepts*, *Hierarchies*, *Relations* and *Links* [80].

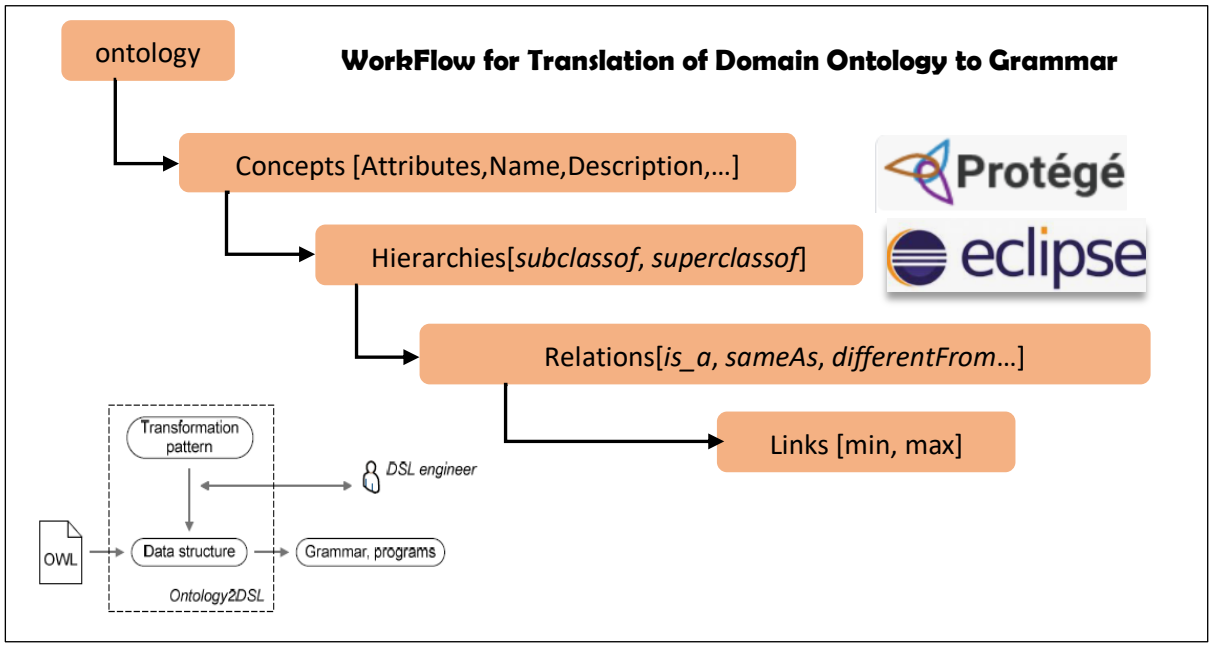


Fig. 77. Overview of Workflow for Future Work.

- Implementation of an *OWLAPI* module for parsing the given interface ontologies, so that it can process various ontology formats. The result that is expected from this module is an *Ontology Object* (OO), that would contain all the information about the *concepts* (classes) and *relations* (hierarchies and properties) extracted from the ontology description contained in the input ontology file.
- Use of an existing *Code Generator* module is a recursive function that traverses the interface *OO* internal structure and visits all the *Concepts* and *Relations* (both hierarchical and non-hierarchical) to generate code in languages like YAML, *etc.*

Based on the presented workflow for translation of API ontologies to corresponding AntLR grammar (in Fig. 77) and literature guidance on WSDL and OWL-S standards in semantic Web domain [90], the current research scope on semantic interoperability of SWCs' API metamodels would be extended towards design of a more concrete, generalized, standardized vehicle domain service API modeling template (including semantic and syntactic specification) for supporting future vehicle domain complex and novel usecases on IoT solutions [79]. With this perspective, the future progress of the presented contribution would also focus on:

- To widen the spectrum for semantic analysis to explore synergies in vehicle service APIs' concepts for semantic interoperability, coverage of more vehicle domain cross-enterprise SOA frameworks' APIs ontological models.
- To ease the accessing of vehicle services APIs semantic data in OWL2 ontology based knowledge graphs for precise service discovery, mapping of the API ontologies (using OWL2) entities in future to widely used OpenAPI standard template (using YAML/JSON schema language definition) in perspective of vehicle domain IoT solutions for future complex automotive usecases [21][88][90].

Chapter 10 Appendix

TABLE XVI. SEMANTIC MAPPING OF INTERFACE FUNCTIONAL TRAITS BETWEEN CROSS-DOMAIN IDLS USING SWC METAMODELS SPECIFICATIONS

Framework	AUTOSAR Adaptive	Franca (including Franca+)	Android	ROS	AUTOSAR Classic
Abstraction of Software Component (SWC) Type	AdaptiveApplicationSwComponent	Component	Service	Node	SwComponent type, CompositionSwCompo-nentType
Provider and Required Interface_ connection_ point	PPortPrototype and RPortPrototype	AnswerMePort, AskMePort	Absent Ports. Input and output messages	Absent Ports. Input and output messages	PPortPrototype , RPortPrototype and PRPortPrototype
Operation based Interface	ServiceInterface	interface	IBinder	Service (command semantics)	PortInterface(ClientServer Interface)
Software Connectors	Absent	delegate, optional	Absent	Absent. Presence of Master node	AssemblySwConnector , DelegationSwConnecto-r
Method Invocation	methods (Client-Server Operation, fireAndForget)	Method call (normal, FireAndForget)	Method call (OnServiceConnected)	Method call (Publish-Subscribe, Client Server)	Client-Server Operation
Data passing Interface	ServiceInterface	Broadcast	ContentProvid-erInterface using URI	Data (Data semantics)	SenderReceiver, NvData, Parameter Interface
VariableData Prototype	Events	NotificationData-element	JobScheduler	Absent	DataElements

TABLE XVII. SEMANTIC MAPPING OF FUNCTIONAL TRAITS BETWEEN CROSS-DOMAIN IDLS USING API CODE SPECIFICATIONS

Framework	Data Types (Primitive and Complex)	Interaction types/Method_ behaviour	Types of Method calls	Connection /association of interfaces	In/Out Argument Types
AUTOSAR Adaptive	Int,float,double, String,Boolean, Associate Map,enum,Vector	Synchronous, Asynchronous	Method (Client-Server Operation: normal, FirenForget)	Event Subscription	InArgumentDataProt-otype, OutArgumentDataPrototype, InOutArgumentDataP-rototype
Franca+	Int, float, double, String, Array, Bytebuffer, enumeration, Boolean, struct, union, Map,constants	Synchronous, Asynchronous	Publish-Subscribe (firenForget), Client-Server(normal)	delegate provide, delegate require, optional provide, optional require	in, out

Android	Int, long, float, char, boolean, double,String, List, Charsequence	Synchronous, Asynchronous	Client-Server	binder (ServiceConn-ection)	in, out and inout
ROS	Int, float, string, constant, Arrays, Boolean, time, duration	Synchronous, Asynchronous	Client-Server, Publish-Subscribe	Topics Subscription	rosparam set, rosparam get
AUTOSAR Classic	Int, float, String, Byte Array, enum, Boolean	Synchronous, Asynchronous (mostly)	Client-Server Operation	PortInterface Mapping, DataType mappings	InArgumentDataPro t-otype, OutArgumentDataP rotoype, InOutArgumentDat aP-rototype

SPARQL Queries for Semantic Mapping of Interface Ontologies (using Protegé 5.5.0 ODE):

}

------OR **Data-Passing:Sender Receiver**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX foaf:<http://xmlns.org/foaf/0.1>

PREFIX f: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#>

PREFIX t: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-23#>

PREFIX k: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-49#>

SELECT ?subclass ?superclass

WHERE { ?subclass (owl:equivalentClass|^owl:equivalentClass)* ?superclass .

filter (?subclass=<http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#AR:SenderReceiverInterface>)

}

ROS Topic:Pub sub

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX foaf:<http://xmlns.org/foaf/0.1>

PREFIX f: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#>

PREFIX t: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-23#>

PREFIX k: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-49#>

PREFIX m: <http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#>

PREFIX p: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#>


SELECT ?subclass ?superclass

        WHERE { ?subclass (owl:equivalentClass|^owl:equivalentClass)* ?superclass .

filter (?subclass=<http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#ROS:Topic>)

}
```

Instance Level Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX foaf:<http://xmlns.org/foaf/0.1>

PREFIX f: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#>

PREFIX t: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-23#>

PREFIX k: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-49#>

PREFIX m: <http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#>

PREFIX p: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#>


SELECT DISTINCT ?individual ?subclass ?superclass
```

```
WHERE {


 ?individual (owl:sameAs|^owl:sameAs)
<http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#DM:FIRE-And-
FORGET> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}
```

----------UNION (Correct)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX foaf:<http://xmlns.org/foaf/0.1>

PREFIX f: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#>

PREFIX t: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-23#>

PREFIX k: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-49#>

PREFIX m: <http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#>

PREFIX p: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#>


SELECT *

WHERE {

{

 ?individual (owl:sameAs|^owl:sameAs)
<http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#ROS:Pub_Sub> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}
```

```
UNION

{

 ?individual (owl:sameAs|^owl:sameAs)
<http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-
35#FNORMAL_METHOD> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}

}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX foaf:<http://xmlns.org/foaf/0.1>

PREFIX f: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#>

PREFIX t: <http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-23#>

PREFIX k: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-49#>

PREFIX m: <http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#>

PREFIX p: <http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#>

SELECT *

WHERE {

{

 ?individual (owl:sameAs|^owl:sameAs)*
<http://www.semanticweb.org/uids2837/ontologies/2020/4/untitled-ontology-57#ROS:Pub_Sub> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}

UNION

{
```

?individual (owl:sameAs|^owl:sameAs)*
<http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#FNORMAL_METHOD> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}

UNION

{

?individual (owl:sameAs|^owl:sameAs)*
<http://www.semanticweb.org/uids2837/ontologies/2020/2/untitled-ontology-35#AR:Request_Response> .

?individual rdf:type ?subclass .

?subclass rdfs:subClassOf ?superclass .

}

}

//

Ontology Mediator: Mediator.owl ontology (R_O) (using Protegé 5.5.0 ODE):

```
<!--
  ////////////////////////////////////////////////////////////////////////////////
  //
  // Annotation properties
  //
  ////////////////////////////////////////////////////////////////////////////////
   -->
  <! -- http://purl.org/dc/elements/1.1/creator -->


  <owl:AnnotationProperty rdf:about="http://purl.org/dc/elements/1.1/creator"/>
     <! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#has_Relation -->
  <owl:AnnotationProperty rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#has_Relation">
     <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">uids2837</dc:creator>
```

```
    <untitled-ontology-321: has_Relation
rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
32#ClientServerOperation"/>

  </owl:AnnotationProperty>

    <! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#part_of -->


  <owl:AnnotationProperty rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-
ontology-32#part_of">

    <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">uids2837</dc:creator>

    <rdfs:comment rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-
ontology-32#Composition_SwComponentType"/>

  </owl:AnnotationProperty>


<!--

  ///////////////////////////////////////////////////////////////////////////////////

  //

  // Classes

  //

  ///////////////////////////////////////////////////////////////////////////////////

   -->

  </owl:Class>

    <! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
32#DM:CommunicationConnectorsSubTypesEnum -->


  <owl:Class rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
32#DM:CommunicationConnectorsSubTypesEnum">

    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-
ontology-32#DM:Composition_SwComponentType"/>

  </owl:Class>

    <! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
32#DM:ComponentPortGroup -->

  <owl:Class rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-
32#DM:ComponentPortGroup">

    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-
ontology-32#DM:PortPrototype"/>

    <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">uids2837</dc:creator>

  </owl:Class>
```

<! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:Composition_SwComponentType -->

<owl:Class rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:Composition_SwComponentType">

<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:SWComponentType_spec"/>

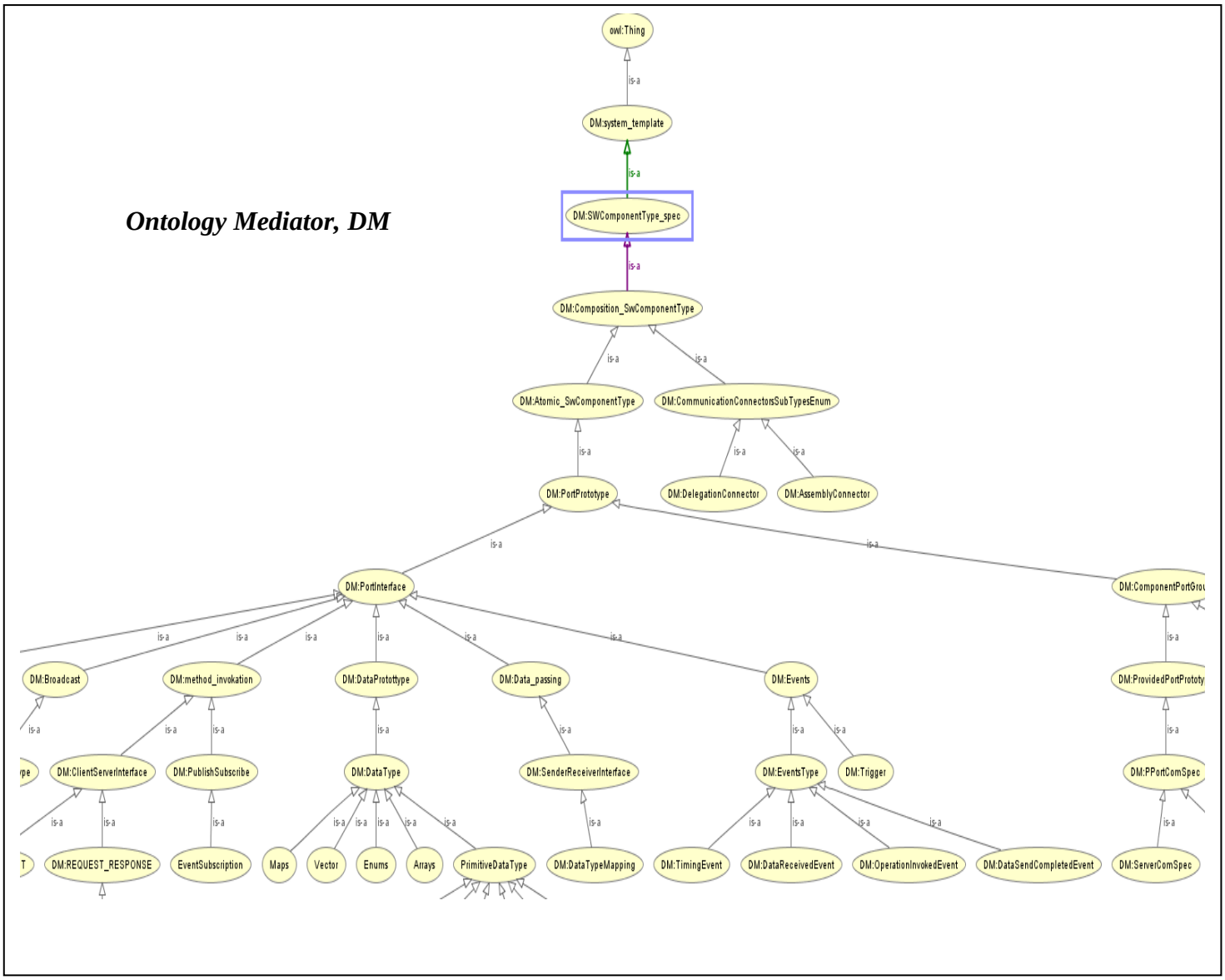
</owl:Class>

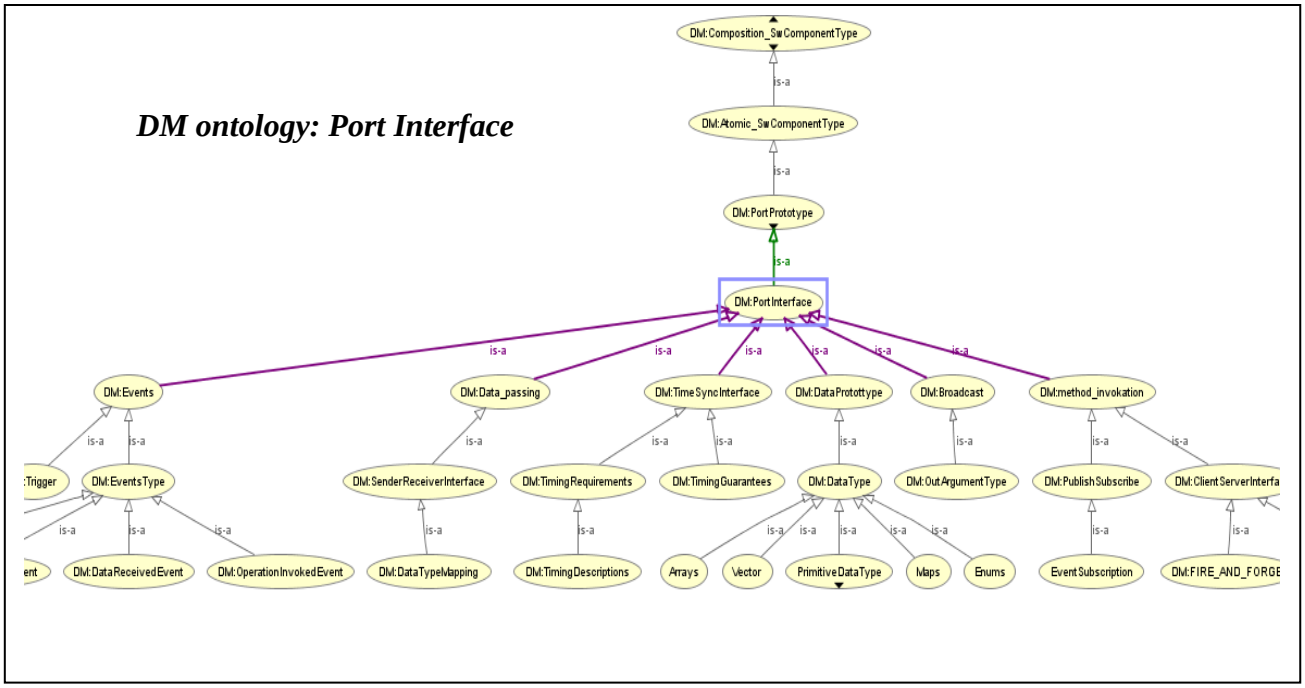
<! -- http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:DataPrototype -->

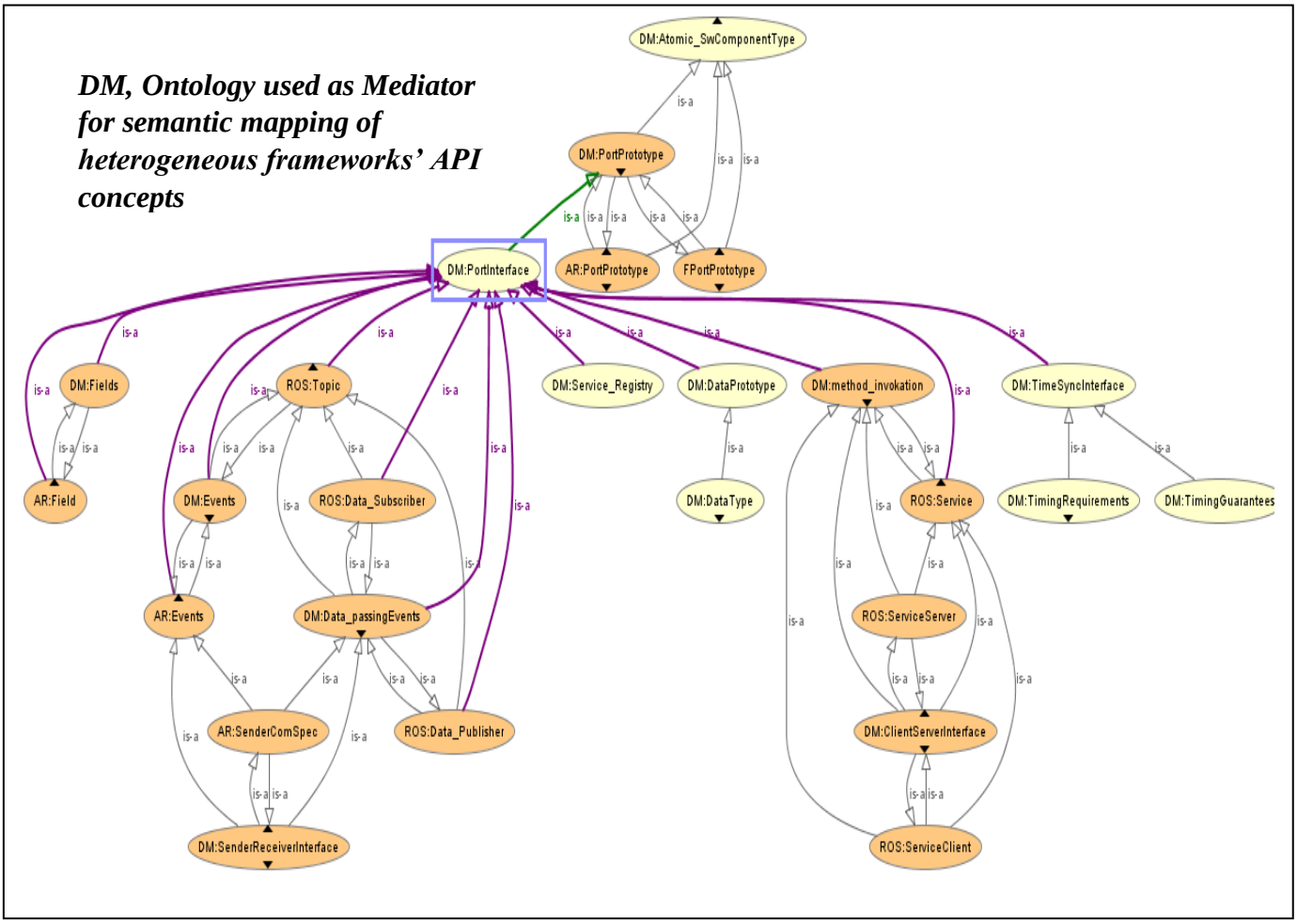
<owl:Class rdf:about="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:DataPrototype">

<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/uids2837/ontologies/2020/0/untitled-ontology-32#DM:PortInterface"/>

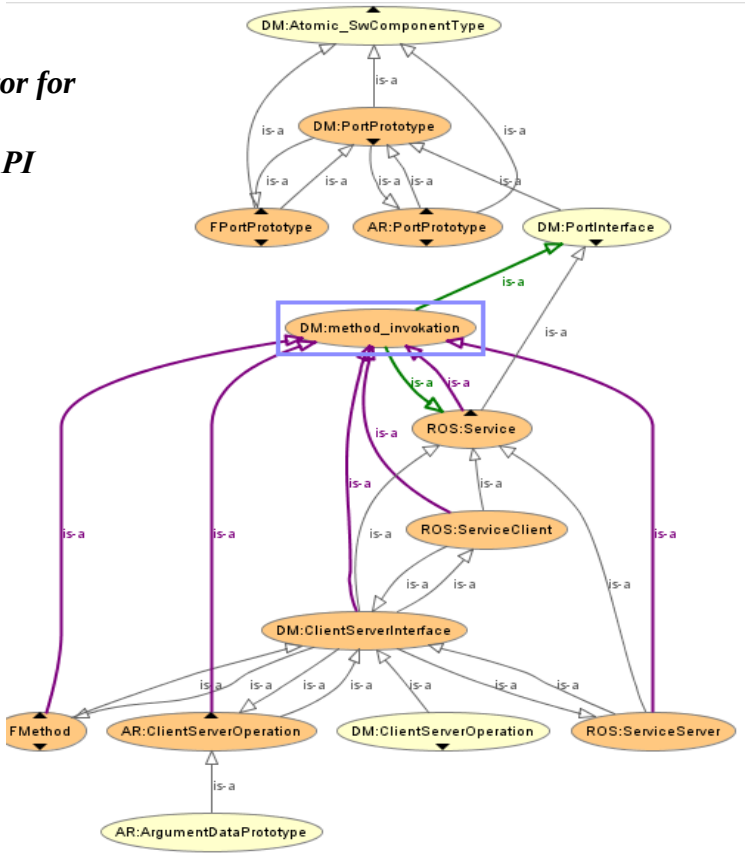
</owl:Class>



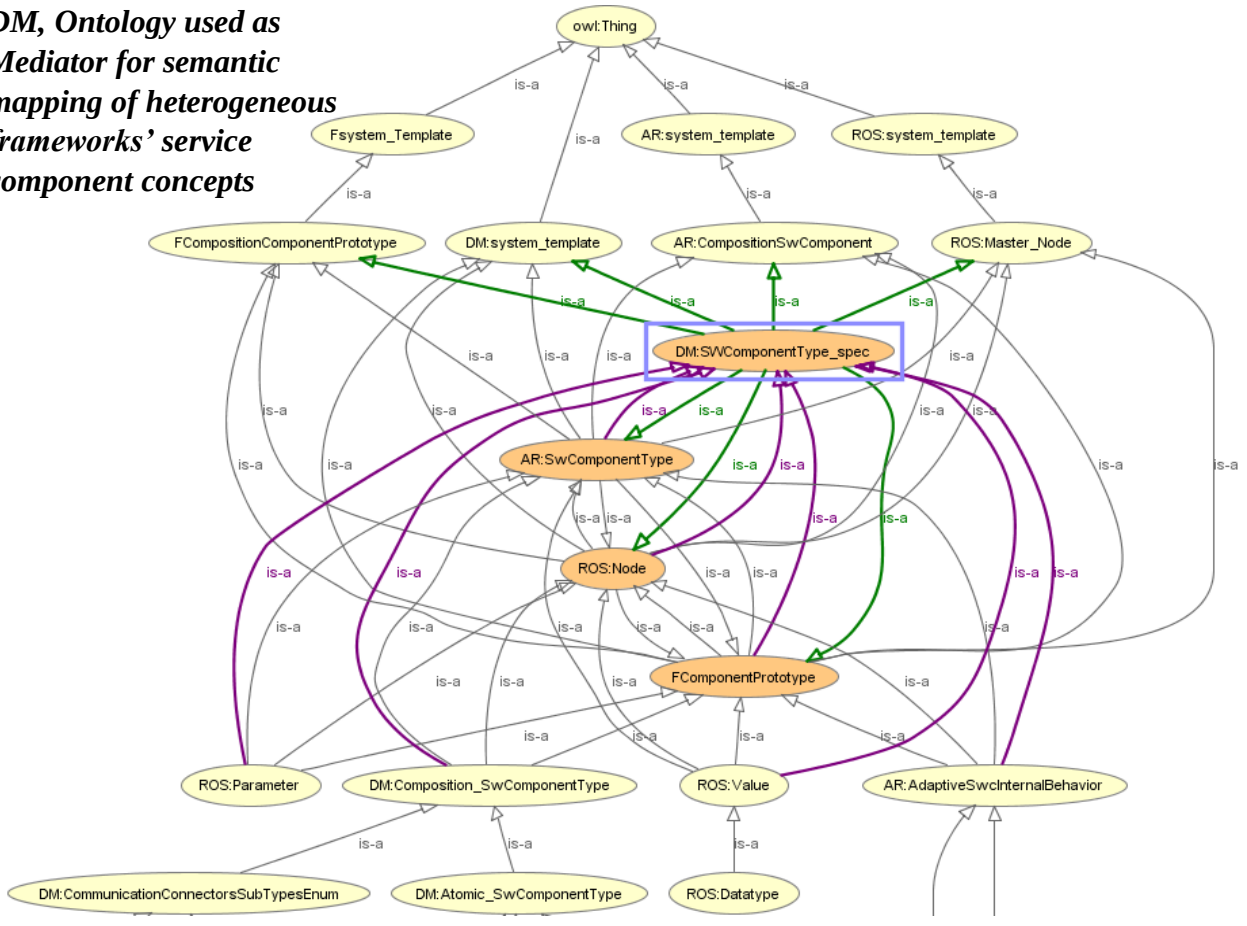

DM ontology: Port Interface
DM:Composition_SwComponentType
DM:Atomic_SwComponentType
DM:PortPrototype
DM:PortInterface
DM:Events
DM:Data_passing
DM:TimeSyncInterface
DM:DataPrototype
DM:Broadcast
DM:method_invokation
:Trigger
DM:EventsType
DM:SenderReceiverInterface
DM:TimingRequirements
DM:TimingGuarantees
DM:DataType
DM:OutArgumentType
DM:PublishSubscribe
DM:ClientServerInterfa
ent
DM:DataReceivedEvent
DM:OperationInvokedEvent
DM:DataTypeMapping
DM:TimingDescriptions
Arrays
Vector
PrimitiveDataType
Maps
Enums
EventSubscription
DM:FIRE_AND_FORGE
is-a


DM, Ontology used as Mediator for semantic mapping of heterogeneous frameworks' API concepts
DM:Atomic_SwComponentType
DM:PortPrototype
DM:PortInterface
AR:PortPrototype
FPortPrototype
DM:Fields
ROS:Topic
DM:Service_Registry
DM:DataPrototype
DM:method_invokation
DM:TimeSyncInterface
AR:Field
DM:Events
ROS:Data_Subscriber
DM:DataType
ROS:Service
DM:TimingRequirements
DM:TimingGuarantees
AR:Events
DM:Data_passingEvents
ROS:ServiceServer
AR:SenderComSpec
ROS:Data_Publisher
DM:ClientServerInterface
DM:SenderReceiverInterface
ROS:ServiceClient
is-a

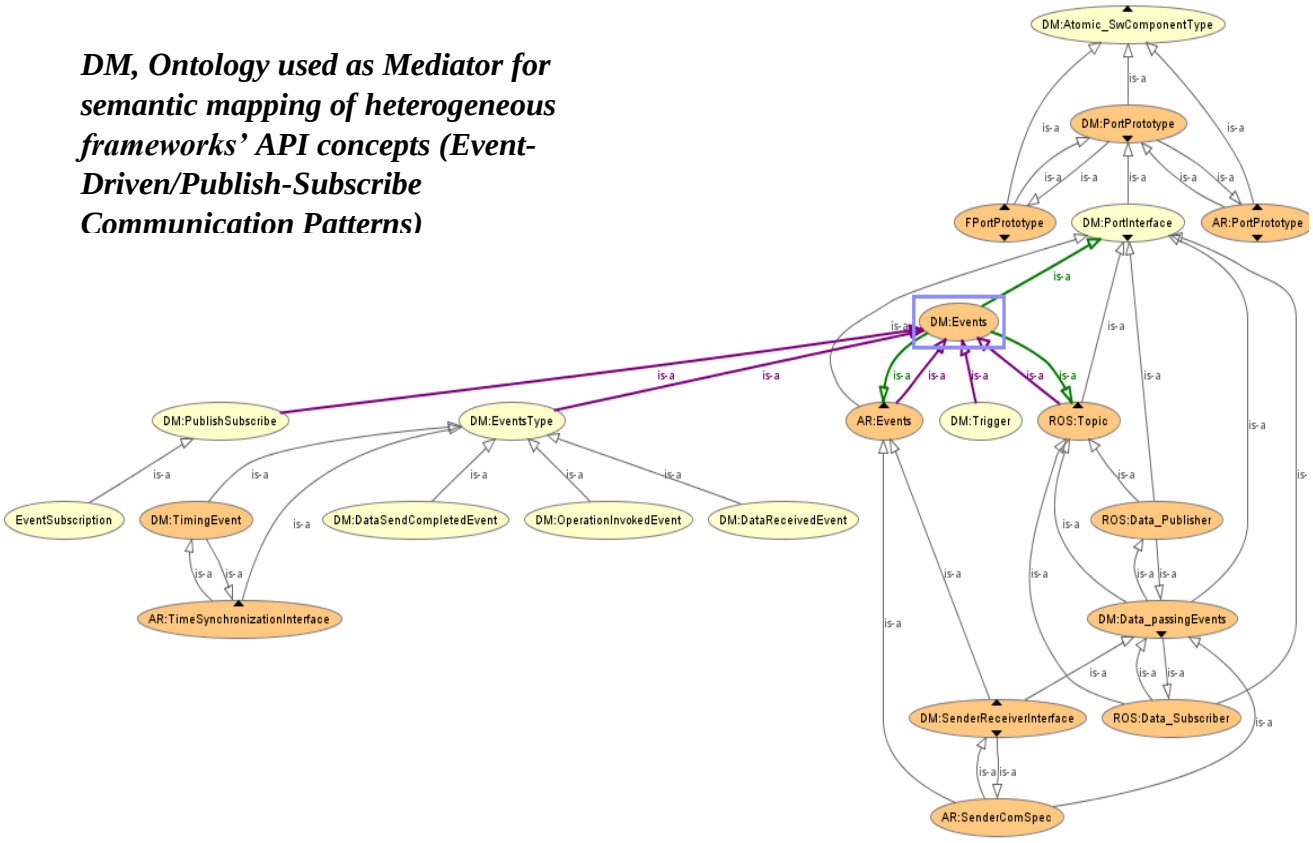
DM, Ontology used as Mediator for semantic mapping of heterogeneous frameworks' API concepts (RPC based method invocation Communication Patterns)



DM, Ontology used as Mediator for semantic mapping of heterogeneous frameworks' service component concepts



DM, Ontology used as Mediator for semantic mapping of heterogeneous frameworks' API concepts (Event-Driven/Publish-Subscribe Communication Patterns)



References

[1] F.S. Parreiras, S. Staab, “Using Ontologies with UML class-based modeling: The TwoUse Approach”, Elsevier Journal on Data and Knowledge Engineering 69(11), 1194-1207, 2010.

[2] F.S. Parreiras, “Marrying Model-Driven Engineering and Ontology Technologies: The TwoUse Approach”, in Wiley Online Library on the Semantic Web and Model-Driven Engineering, pp. 44-59, 2012.

[3] S. Staab, T. Walter, G. Gröner, F.S. Parreiras, “Model Driven Engineering with Ontology Technologies”, In: Aßmann U., Bartho A., Wende C. (eds) Reasoning Web. Semantic Technologies for Software Engineering. Reasoning Web. Lecture Notes in Computer Science, vol 6325. Springer, Berlin, Heidelberg, 2010.

[4] K. Arnarsdóttir, A.-J. Berre, A. Hahn,M. Missikoff, F. Taglino, “Semantic mapping: ontology-based vs. model-based approach RoAlternative or complementary approaches? In: Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability, Luxembourg, 2006.

[5] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, M. Wimmer, “Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages,” In: Nierstrasz O., Whittle J., Harel D., Reggio G. (eds) MODELS, Springer Lecture Notes in Computer Science, vol 4199. pp. 528-542, 2006.

[6] C. Paniagua, J.Delsing and J.Eliasson, “Interoperability Mismatch Challenges in Heterogeneous SOA-based Systems”, IEEE International Conference on Industrial Technology (ICIT), DOI:10.1109/ICIT.2019.8754991, 2019.

[7] S. De, M. Niklas, B. Rooney, J. Mottok, P. Brada, “Semantic Mapping of Component Framework Interface Ontologies for Interoperability of Vehicle Applications”, Elsevier Procedia Computer Science. 170. 813-818. 10.1016/j.procs.2020.03.151, 2020.

[8] S. De, M. Niklas, B. Rooney, J. Mottok, P. Brada, “Towards Semantic model-to-model Mapping of Cross-Domain Component Interfaces for Interoperability of Vehicle Applications An Approach towards Synergy Exploration”, In: CEUR Workshop proceedings, ModComp, Vol. 2442, Munich Germany, 2019.

[9] S. Staab, T. Walter, F.S. Parreiras, “An ontology-based framework for domain-specific modeling”. Softw Syst Model 13, 83–108 https://doi.org/10.1007/s10270-012-0249-9, 2014.

[10] T. Pramsohler, S. Schenk, A. Barthels und U Baumgarten. “A layered interface-adaptation architecture for distributed component-based systems”. In: Future Generation Computer Systems,Elsevier,Vol 47,June 2015,pp 113-126.

[11] B. Motik, P.F. Patel-Schneider, I. Horrocks, “ OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax.http://www.w3.org/TR/owl2-syntax/ Accessed Oct 2009.

[12] H. Elasri,,E.Elabbassi,S.Abderrahim and Muhammad, “Semantic integration of UML Class diagram with Semantic Validation on Segments of Mapping”,ArXiv 2018.

[13] S. De, M. Niklas, J. Mottok and P. Brada, “Semantic Synergy Exploration in Interface Description Models of Heterogeneus Vehicle Frameworks: Towards Automotive Meta Interface Description Model”, ARCS Workshop 2019; 32nd IEEE International Conference on Architecture of Computing Systems, Copenhagen, Denmark, 2019, pp. 1-8.

[14] S. De, M. Niklas, J. Mottok, and P. Brada, :”A Semantic Analysis of Interface Description Models of Heterogeneous Vehicle Application Frameworks: An Approach Towards Synergy Exploration”. DOI: 10.5220/0007472503940401,In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pages 394-401,ISBN: 978-989-758-358-2.

[15] S. De, M. Niklas, J. Mottok and P. Brada, “Model Transformation of Application Software Component from Classic to Adaptive AUTOSAR: An Approach to Migrate Software Components”, 44th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) Workshop, Prague, Czech Republic, August 29th - 31st, 2018.

[16] Birken, K., http://www.bmw.com “Franca User Guide”, “Franca Component Definition language Franca+ User guide” “.Release 0.12.0.1, Eclipse Foundation, itemis AG, 2013. Release 0.13.0, BMW Group,2018.

[17] F. Jiménez, J. E. Naranjo, J. J. Anaya, F. García, A. Ponz, J. M. Armingol, Advanced Driver Assistance System for Road Environments to Improve Safety and Efficiency, Transportation Research Procedia, vol. 14, pp.2245-2254, 2016.

[18] S. De, M. Niklas, B. Rooney, J. Mottok and P. Brada, “Towards Translation of Semantics of Automotive Interface Description Models from Franca to AUTOSAR Frameworks : An Approach using Semantic Synergies”, 2019 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 2019, pp. 1-6, doi: 10.23919/AE.2019.8867018.

[19] A free, open-source ontology editor and framework for building intelligent systems, https://protege.stanford.edu/ .

[20] R. Weinreich, and J. Sametinger, “Component models and component services: Concepts and principles”, G.T. Heineman and W.T. Councill (eds.), Reading, MA: Addison-Wesley, 2001 pp. 33–48.

[21] Shearer, Rob. “Structured Ontology Format”, 2007.

[22] A. Goknil, J. Suryadevara, MA. Peraldi-Frati, F. Mallet: “Analysis Support for TADL2 Timing Constraints on EAST-ADL Models. In: Drira K. (eds) Software Architecture”. ECSA 2013. Lecture Notes in Computer Science, vol 7957. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39031-9_8, 2013.

[23] Object Management Group (OMG): Ontology Definition Metamodel. Version 1.1, http://www.omg.org/spec/ODM/1.1 (2014).

[24] S. Brockmans, R. Volz, A. Eberhart, P. Löffler,”Visual Modeling of OWL DL Ontologies Using UML”, In: McIlraith S.A., Plexousakis D., van Harmelen F. (eds) The Semantic Web – ISWC. Lecture Notes in Computer Science, vol 3298. Springer, Berlin, Heidelberg, 2004.

[25] T. Rahmani, D. Oberle, M. Dahms : “An Adjustable Transformation from OWL to Ecore”. In: Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part II. Volume 6395 of Lecture Notes in Computer Science., Springer.243-257, 2010.

[26] OWLizer. http://st.inf.tu-dresden.de/owlizer/ (2016).

[27] T.Wang, S.Truptil and F.Benaben, “An automatic model-to-model mapping and transformation methodology to serve model-based systems engineering”, Information Systems and EBusiness Management, Springer Verlag, 2017, 15 (2, SI), pp.323-376.

[28] D.Di. Ruscio, D. Wagelaar, L. Iovino and A.Pierantonio "Translational Semantics of a co-evolution Specific language with the EMF Transformation Virtual Machine", ICMT 2012, pp 71-89.

[29] D. Bálek, F. Plášil: "Software Connectors and Their Role in Component Deployment". (eds) New Developments in Distributed Applications and Interoperable Systems.DAIS 2001. IFIP International Federation for Information Processing, vol 70. Springer, Boston, MA, 2001.

[30] H. Bruyninckx, N. Hochgeschwender, L. Gherardi, M. Klotzbücher, G.Kraetzschmar, D. Brugali, "The BRICS Component Model: a Modelbased Development Paradigm for Complex Robotics Software Systems", Annual ACM Symposium on Applied Computing (SAC).

[31] A. G. Parada and L. Brisolara, "A Model Driven Approach for Android Application Development", Brazilian Symposium on Computing System Engineering, 2012.

[32] S. Chris, T. Martin and V. Markus: "Model-based Middleware for Embedded Systems". INFORMATIK 2004 - Informatik verbindet, Band 2, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Ulm, 20.-24. September 2004.

[33] S. Jafar, B. Chhaya, U. Durak: "OWL Ontology to Ecore Metamodel Transformation for Designing a Domain Specific Language to Develop Aviation Scenarios", Mod4Sim '17, pp.1-11, VA, USA (2017).

[34] G.Hillairet: Eclipse Modeling for Semantic Web. https://github.com/ghillairet/emf4sw, https://www.eclipse.org/atl/usecases/ODMImplementation/. (2007).

[35] Domingue J., Fensel D., & Roman, D.: Semantic Web Services with the Web Service Modeling Ontology (WSMO), pp.7-9, (2005).

[36] N., Lê & Y. Feng, R. Seungmin and K. Rajaraman: "Enabling Interoperability across Heterogeneous Semantic Web Services with OWL-S Based Mediation". Proceedings - 2011 IEEE Asia-Pacific Services Computing Conference, APSCC 2011. 471-476. 10.1109/APSCC.2011.78. (2011).

[37] H. Blom, D. Chen,H. Kaijser, H. Lönn,Y. Papadopoulos,M.O. Reiser, R.T. Kolagiri, and S.Tucci: "EAST-ADL:An Architecture Description Language for Automotive Software-intensive Systems in the Light of Recent use and Research",IJSDA,vol.5,no. 3, pp-1-20, 2016.

[38] I. Crnkovic, M. Larsson, "Building Reliable Component-Based Software Systems", First Edition, ARTEC HOUSE, INC.,2002.

[39] C. Szyperski, D. Gruntz and S. Murer, " Component Software Beyond Object-Oriented Programming", Second Edition, ACM press, 2002.

[40] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K.Wallnau: "Technical concepts of component-based software engineering", In:Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute, Volume II, (pp. 26-29), 2000.

[41] AUTOSAR, "Specification of Manifest", AUTOSAR AP Release 18-10, 2017.http://www.autosar.org.

[42] AUTOSAR, http://www.autosar.org, "Integration of Franca IDL SWC Descriptions", AUTOSAR Release 16-11,November 2016.

[43] AUTOSAR, http://www.autosar.org, "SWC Template", "Virtual Function Bus",AUTOSAR CP Release 4.3.0. May, 2016.

[44] N. Medvidovic and R.N. Taylor: "A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering", 2000, 26, 70-93.

[45] I.Crnkovic, S.Sentilles, A.Vulgarakis and M.Chaudron, "A Classification Framework for Component Models", IEEE Transactions on Software Engineering 37 (5), 593-615.

[46] C. Szyperski, D. Gruntz and S. Murer, " Component Software Beyond Object-Oriented Programming", Second Edition, ACM press,2002.

[47] S. Brockmans, P. Haase and R. Studer: "A MOF-based Metamodel and UML Syntax for Networked Ontologies", 2006.

[48] N. Nassar, T. Arendt and G. Taentzer: "Deriving Model Metrics from Meta Models", Conference: http://dblp.org/rec/conf/modellierung/NassarAT16 ,Volume: Vol P-254 (2016).

[49] J.R Williams, A. Zolotas, N.D. Matragkas, L.M. Rose, D.S. Kolovos, R.F. Paige, F.A. Polack: "What do metamodels really look like" Eessmod@ Models 1078, 55-60 (2013).

[50] J. Di Rocco, D. Di Ruscio, L. Iovino, A. Pierantonio: "Mining Metrics for Understanding Metamodel Characteristics", In: Proceedings of the 6th International Workshop on Modeling in Software Engineering. pp. 55{60. MiSE 2014, ACM, New York, NY, USA (2014).

[51] D. Bork: "Metamodel-Based Analysis of Domain-Specific Conceptual Modeling Methods. In: Buchmann R., Karagiannis D., Kirikova M. (eds) The Practice of Enterprise Modeling. PoEM 2018. Lecture Notes in Business Information Processing, vol 335. Springer, Cham. https://doi.org/10.1007/978-3-030-02302-7_11, (2018).

[52] J. Mylopoulos: "Conceptual modelling and Telos",In: Loucopoulos, P., Zicari, R. (eds.) Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development, New York: John Wiley & Sons. pp. 49-68 (1992).

[53] S. Tartir, I. Arpinar and A. Sheth: "Ontological Evaluation and Validation. 10.1007/978-90-481-8847-5_5,(2010).

[54] H. Alani, C. Brewster and N. Shadbolt: "Ranking Ontologies with AKTiveRank", In: 5th International Semantic Web Conference. November, 5-9, 2006.

[55] A. Lozano-Tello and A. Gomez-Perez: "ONTOMETRIC: a method to choose the appropriate ontology", Journal of Database Management 2004, 15, 1-18.

[56] N. Guarino, and C. Welty: "An Overview of OntoClean", Staab, S. and Studer, R. eds. Handbook on Ontologies, Springer Verlag, 2004, 151-159.

[57] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann: "Modelling Ontology Evaluation and Validation", In proceedings of the 2006 European Semantic Web Conference, 2006.

[58] E. Paslaru, B. Simperl, C. Tempich, Y. Sure: "ONTOCOM: A Cost Estimation Model for Ontology Engineering", In: Proceedings of fifth International Semantic Web Conference (ISWC 2006), Athens, GA, USA, 2006.

[59] T. Bosch and B. Mathiak, "XSLT transformation generating OWL ontologies automatically based on XML Schemas," 2011 International Conference for Internet Technology and Secured Transactions, Abu Dhabi, 2011, pp. 660-667.

[60] P.F. Patel-Schneider, and J. Simeon , "The Yin/Yang web: XML syntax and RDF semantics". in 11th International World Wide Web Conference (WWW2002),pp.443–453, 2002.

[61] J. Ferreira, "VCO-vehicle corporate ontology", in 2nd Conferencia Ibérica de Sistemas e Tecnologias de Informacia,pp.21-23, Porto, Potugal, 2007.

[62] F.Cruz Isabel, H. Xiao, and F. Hsu,"An Ontology-based Framework for XML Semantic Integration", in Database Engineering and Applications Symposium. IDEAS'04. Proceedings. International. IEEE, pp. 217—226, 2004.

[63] C. Tsinaraki and S. Christodoulaki, "XS2OWL: A Formal Model and a System for enabling XML Schema Applications to interoperate with OWLDL Domain Knowledge and Semantic Web Tools", in International DELOS Conference,pp. 124- 136, Pisa, Italy, 2007.

[64] H. Afzal, M. Waqas, and T. Naz, "OWLMap: Fully Automatic Mapping of Ontology into Relational Database Schema", International Journal of Advanced Computer Science and Applications , Vol. 7(11), 2016.

[65] Top Braid composer, https://www.topquadrant.com/ .

[66] N. Noy, D. McGuinness and P.Hayes, "Semantic Integration and Interoperability Using RDF and OWL", 2005. www.w3.org.

[67] P. Heyvaert, A. Dimou and A. Herregodts and Verborgh, R. Verborgh, D. Schuurman, E. Mannens and R. V. d. Walle1 , "RMLEditor: A Graph-based Mapping Editor for Linked Data Mappings",pp. 709-723. DOI:10.1007/978-3-319-34129-3_43, 2016.

[68] T. Pramsohler and U. Baumgarten, "Adaptation of automotive infotainment interfaces using static and dynamic adapters", In: Horbach, M. (Hrsg.), INFORMATIK 2013 – Informatik angepasst an Mensch, Organisation und Umwelt. Bonn: Gesellschaft für Informatik e.V.. (S. 2488-2501), 2013.

[69] C. Berger and M. Dukaczewski, "Comparison of Architectural Design Decisions for Resource-Constrained Self-Driving Cars-A Multiple Case-Study", in Gesellschaft for Informatik 2014.

[70] H. Benouda, R. Essbai, M. Azizi and M. Moussaoui, " Modeling and Code Generation of Android Application Using Acelo", International Journal of Software Engineering and Its Applications vol. 10 (3), pp. 83-94, 2016.

[71] A.Shakhimardanov, N.Hochgeschwender, and G. K. Kraetzschmar, "Component Models in Robotics Software". In Proceedings of the Performance Metrics for Intelligent Systems Workshop (PerMIS 2010). Baltimore, US.A., 2010.

[72] Dhama, "Interface Definition Languages", EB white paper, May 2017, http://www.elektrobit.com.

[73] M. Slee, A. Agarwal and M. Kwiatkowski, "Thrift: Scalable Cross-language Services Implementation", Facebook white paper 5, 156 university ave, CA, April 2007.

[74] Documenting Your Existing APIs: API Documentation Made Easy with OpenAPI & Swagger, https://swagger.io/resources/articles/documenting-apis-with-swagger/ .

[75] RobMoSys,"Block-Port.Connector", RobMoSys Wiki, http://www.robmosys.eu. June 2017.

[76] D. Vrandecic, and Y. Sure, "How to design better ontolo-gy metrics. In Proceedings of the 4th European Semantic Web Conference" (ESWC'07) 4519: 311–325, Springer, 2007.

[77] Y. Hu and I. Neamtiu, "Static detection of event-based races in android apps", Proceedings of the Twenty- Third International Conference on Architectural Support for Programming Languages and Operating Systems ser. ASPLOS '18, pp. 257-270, 2018.

[78] Callbacks and Spinning, www.ros.org .

[79] A. Ojamaa, HM. Haav and J. Penjam, "Semi-automated Generation of DSL Meta Models from Formal Domain Ontologies", In: Bellatreche L., Manolopoulos Y. (eds) Model and Data Engineering. Lecture Notes in Computer Science, vol 9344. Springer, Cham. https://doi.org/10.1007/978-3-319-23781-7_1 , 2015.

[80] Fonseca, J.M.S. & Pereira, Maria & Rangel Henriques, Pedro, "Converting ontologies into DSLs", In: OpenAccess Series in Informatics, pp. 85-92. DOI: 10.4230/OASIcs.SLATE.2014.85, 2014.

[81] C. Hu, X. Zhang, Q. Zhao and C. Zhao, "Ontology-Based Semantic Integration Method for Domain-Specific Scientific Data," Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), Qingdao, pp. 772-777, doi: 10.1109/SNPD.2007.431, 2007.

[82] J. Cardoso, "Approaches to Developing Semantic Web Services". World Academy of Science, Engineering and Technology, Open Science Index 14, International Journal of Computer and Information Engineering, 2(2), 611 – 624, 2006.

[83] K. Furdík, M. Tomášek and J. Hreno, "A WSMO-based Framework Enabling Semantic Interoperability in e-Government Solutions", Acta Polytechnica Hungarica,vol.8, No.2, 2011.

[84] Z. Vales and P. Brada, "Service API Modeling and Comparison: A Technology-Independent Approach," 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 2020, pp. 158-161, doi: 10.1109/SEAA51224.2020.00034, 2020.

[85] SWAGGER,"OpenAPI Specification", Version 3.0.3, https://swagger.io/specification/ .

[86] Proposal: Expand OpenAPI to include RPC APIs,"OpenAPI Specification", https://github.com/OAI/OpenAPI-Specification/issues/801, 2016.

[87] E. Kilgarriff, B. Sapkota, L. Vasiliu and D. Aiken, "XML to WSML adapter Implementation", Proceedings of the 2nd WSMO Implementation Workshop, 2005.

[88] P. Espinoza-Arias, Garijo D. and O. Corcho , "Mapping the Web Ontology Language to the OpenAPI Specification", In: Advances in Conceptual Modeling. ER 2020, vol 12584. Springer, Cham, 2020.

[89] A. Heß, E. Johnston, and N. Kushmerick, "ASSAM: A tool for semi-automatically annotating semantic web services,"in Proceedings of the Third International Semantic Web Conference (ISWC), 2004, pp. 320–334.

[90] S. Schwichtenberg, C. Gerth and G. Engels, "From Open API to Semantic Specifications and Code Adapters," 2017 IEEE International Conference on Web Services (ICWS), USA, 2017, pp. 484-491.