Towards Semantic model-to-model Mapping of Cross-Domain Component Interfaces for Interoperability of Vehicle Applications

An Approach towards Synergy Exploration

Sangita De, Michael Niklas and Brian Rooney
Continental AG
{sangita.de,michael.niklas and brian.rooney}@continental-corporation.com

Juergen Mottok
Dept. of Electrical Engineering and Information Technology
OstBayerische Technical University(OTH),Regensburg
Regensburg, Germany
juergen.mottok@oth-regensburg.de

Premek Brada
Dept. of Computer Science and Engineering, Faculty of Applied Sciences
University of West Bohemia
Pilsen, Czech Republic
brada@kiv.zcu.cz

***Abstract*—With the increase in demand of services in the automotive industry, automotive enterprises prefer to collaborate with other qualified cross-domain partners to provide complex automotive functions (or services), such as autonomous driving, OTA (Over The Air) vehicle update, V2X (Vehicle-to-Vehicle communication), etc. One key element in cross-domain enterprise collaboration is the mutual agreement between interfaces of software components. In this context, model-to-model mappings of software component models of heterogeneous frameworks for automotive services and to explore the synergies in their interface semantics, have become an essential factor in improving the interoperability among the automotive and other cross-domain enterprises. However, one of the challenges in achieving cross-domain component interface model-to-model mappings at an application level lies in detecting the interface semantics and the semantic relations that are conveyed in different component models in different frameworks. This paper addresses this challenge using a Model Driven Architecture (MDA) based analytical approach to explore interface semantic synergies in the cross-domain component meta-models that are used for automotive services. The approach applies manual semantic checking measurements at an application interface level to understand the meanings and relations between the different meta-model entities of cross-domain framework software components. In this research, we attempt to ensure that interface description models of software components from heterogeneous frameworks can be compared, correlated and re-used for automotive services based on semantic synergies. We have demonstrated our approach using component meta-models from cross-domain enterprises, that are used for the automotive application domain.**

Keywords—component, Framework, domain, interface, semantic, mapping, synergy, metamodel, syntax, application

I. INTRODUCTION

Today's vehicle electronics are essentially clusters of heterogenous ECUs (Electronic Control Units) from various suppliers with varying levels of complexity. From simple sensor/actuators all the way up to High Performance Computing (HPC) chipsets, communicating over heterogeneous communication networks or even off-car to Wireless Networks. The application (app) developers must have knowledge of a wide range of technologies instead of being focused on a particular technology such as programming or data interchange. The automotive software industry always looked for means to narrow the gap on the way from requirement to software. Therefore, this would require interface adaption [6] at the app component model level to achieve transparent interoperability. Since a component model is much easier to understand and maintain than code, the investment in MDE (Model Driven Engineering) to achieve transparent interoperability among app software components (SWCs) continues to pay back in long-term.

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 Interface Description Language (IDL) defines the software interface agreements between the app component interfaces. IDLs are typically bound to one or more programming language generators. Over the time, in the automotive app 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 SWC 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 FW SWCs and to achieve interoperability by reusing of artifacts, requires understanding of the semantic mapping at app SWC interface level [1][9]. The IDLs that are used for automotive app domain SOSCM (Service- Oriented Software Component Model) interface description may need to consider the following fundamental characteristics[1]:

- *Interface type*: The distinction of the basic interface type: operation-based (e.g. methods invocations) and data-based service interface (e.g. data passing).
- *Separation of Interface Roles:* The distinction between the provides-part and the requires-part of a service interface.
- *Interface interaction points*: Service interface interaction points (e.g. ports, topics, etc.).
- *Method invocation*: Method signatures containing information with valid parameter types, e.g. Client-server, Sender-Receiver, Publish-Subscribe etc.

- *Attributes*: Specification of attributes or fields e.g. getters, setters, Notifiers, etc.
- *Abstraction of Component*: abstract description of the SWC (single or composite) using the interfaces.
- *Interaction with Connectors*: Specification of software connectors used for realization of an interface mapping between provider and receiver SWCs interfaces.
- *Interface Behavior & Semantic Annotation constraints*: The invariants, pre- and postcondition constraints of a component interface internal behavior depends on the state of the SWC.
- *Interface Binding*: The binding type describes the way a vehicle app SWC interfaces binds to a middleware communication protocol for intra- or inter-ECU communication.

A component *construct* fundamentally combines both service interfaces and interaction description. However, SWC interface *binding* to middleware is not considered in the current scope to align the focus towards interface semantic synergies exploration for heterogeneous component models purely at app level.

A. Contribution of the Report

The automotive industry can be regarded as a complex yet connected network (ecosystem) of highly interdependent subsystems as seen in Fig. 1. Systems with a heterogeneous implementations, architectures, semantics have to be integrated into collaborative environments to support automotive complex services. The interoperability between them has become a major challenge in this new ecosystem, thereby generating several research questions about how to manage the information exchange and collaboration between these heterogeneous system's FWs at app level with so vastly different properties [17]. This paper presents a detailed investigation on semantic survey of the component interface models of various cross-domain FWs. The paper explores the possibilities of semantic service-based interfaces matches and reveals the areas of semantic mismatches between the information ex-change formats of heterogeneous system's FWs at an app level where the interoperability is crucial [17]. The proposed solution in this paper is based on exploration of component interface semantic synergies [2][9]. Exploration of interface semantic synergies could also further aid in SWC reusability in the future.

B. Motivation Scenario and Related Work

In the last decade a plethora of different interface specification models or IDLs were designed using different technologies to support automotive app domain. This could be due to the fact: firstly, coexistance of new ECU HW interfaces with legacy as seen in Fig. 1, secondly, the conformance to frequent new standards in automotive domain [10], thirdly, the non-functional system requirements such as performance and scalability. Unlike adaption of ADLs (Architecture Description Languages) of frameworks that requires adaption of the entire end-to-end stack at system specification level, adaption of IDLs would basically focus on adaption of components, ports, connectors, algorithmic code of FW SWCs purely at an app level [18]. To increase interoperability and efficient reuse of component interface model, it is important to understand the differentiation of component model interfaces.

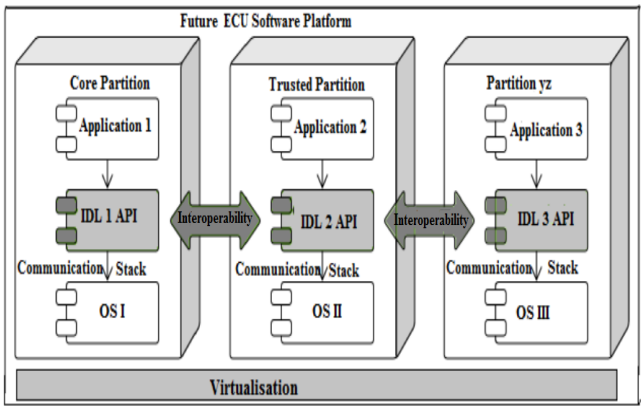

Fig. 1. An Overview of Service Interdependent Communication between the ECU partitions

The authors of [1] proposes a *Component Model Classification* FW which identifies and discusses the basic principles of component models. The authors of [15] 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. To deal with interoperability, 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 cost and test effort. A third option could be to possibly consider the role of connectors in the construction of a software system from reusable components that is to consider especially the role connectors should play when the distribution and deployment of a component-based application is considered, as proposed by authors of [7].

In this context Software Adaption is a promising approach to build flexible interfaces for variable software systems. Authors of [17] proposes adapter systems to deal with service mismatch problems that can happen in the information exchange in heterogeneous SOA-based environments where the interoperability is crucial. The authors of [2] 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 [11] propose component model-to-model transformations to establish translation of semantics by manual mapping of programming languages of heterogeneous platforms [11].

II. METHODOLOGY

With the proposed methodology based on static analysis of interface semantics, we have attempted to provide a level of abstraction at SWC interface semantic specification level and have defined an abstract UML profile (M2 level) model also called Component-Port-Connector (CPC) model [1] [7] [5], to describe each of the cross-domain FW SWC *constructs* in context of *interfaces*. The SWC *constructs* agrees mostly to the traits that are visible in the *Black-box* view of a component. A SWC *construct* of SOA based automotive app domain FWs fundamentally represents (i) the SWC *service-based-interfaces* used for the *interaction* with the other components for inter- or intra-ECU communication, and (ii) the means of SWC *binding* and communication using middleware. With the

given approach each of the FW SWC *constructs* are represented using the CPC models, abstracted from the SWC meta-models of heterogeneous domain FWs.

A. *Classified Levels for Semantic Survey of Software Component Interfaces*

To illustrate the approach of static semantic analysis of SWC interfaces, we have considered few of the SWC *constructs* of the cross-domain platforms supporting automotive apps. In this approach, *Interface Syntactic level* was not considered as it covers only the syntactic aspect and describes the signature of an interface in a FW specific programming language and is relatively out of current research scope. For simplicity, we have classified SWC *constructs* into three basic levels [1][12] :

- *Interface Semantic level*: reinforces the *syntactic* level and concerns with the meaning of features often specified by the expectations and effects of individual features. A generalization of this level can be assumed as semantics [12].
- *Interface Behavioral level*: represents dynamic behavior of service-based interfaces based on constraints (e.g. constraints on their temporal ordering, precondition, postcondition, invariants, etc.). It expresses their internal behavior (e.g. internal states).
- *Composition level*: Connection represents interactions between interface functionalities and behavior in two components as far as accessible through SWC ports [5] e.g. *Synchronous, Asynchronous, etc.*

Fig. 2 illustrates automotive app domain SWC *constructs* represented by an abstract generic CPC model [1][12][19]. Towards the SWC interface semantic synergy exploration, the structure of an abstract generic CPC model illustrates the abstract view of the components at different containment levels, their interface types, their typed input and output ports, and the connectors between them. Abstraction of the generic CPC model emphasize on the common service-based interface semantic properties and hide the platform specific details that are not needed in the interface description [19].

A generic specific CPC model can further provide knowledge base for future automotive domain specific software solution such as coherent, unified IDL or a Meta-IDL model. With the reference to the abstract generic CPC model, we have tried to represent the CPC model for each of the FW SWC *constructs* based on three identified basic levels: *Interface semantic*, *Interface Behavior* and *Composition* [1]. With the semantic survey we have compared and revealed the areas of service interface semantic matching and mismatches among the cross-domain FW IDLs at model level in reference to the generic CPC model.

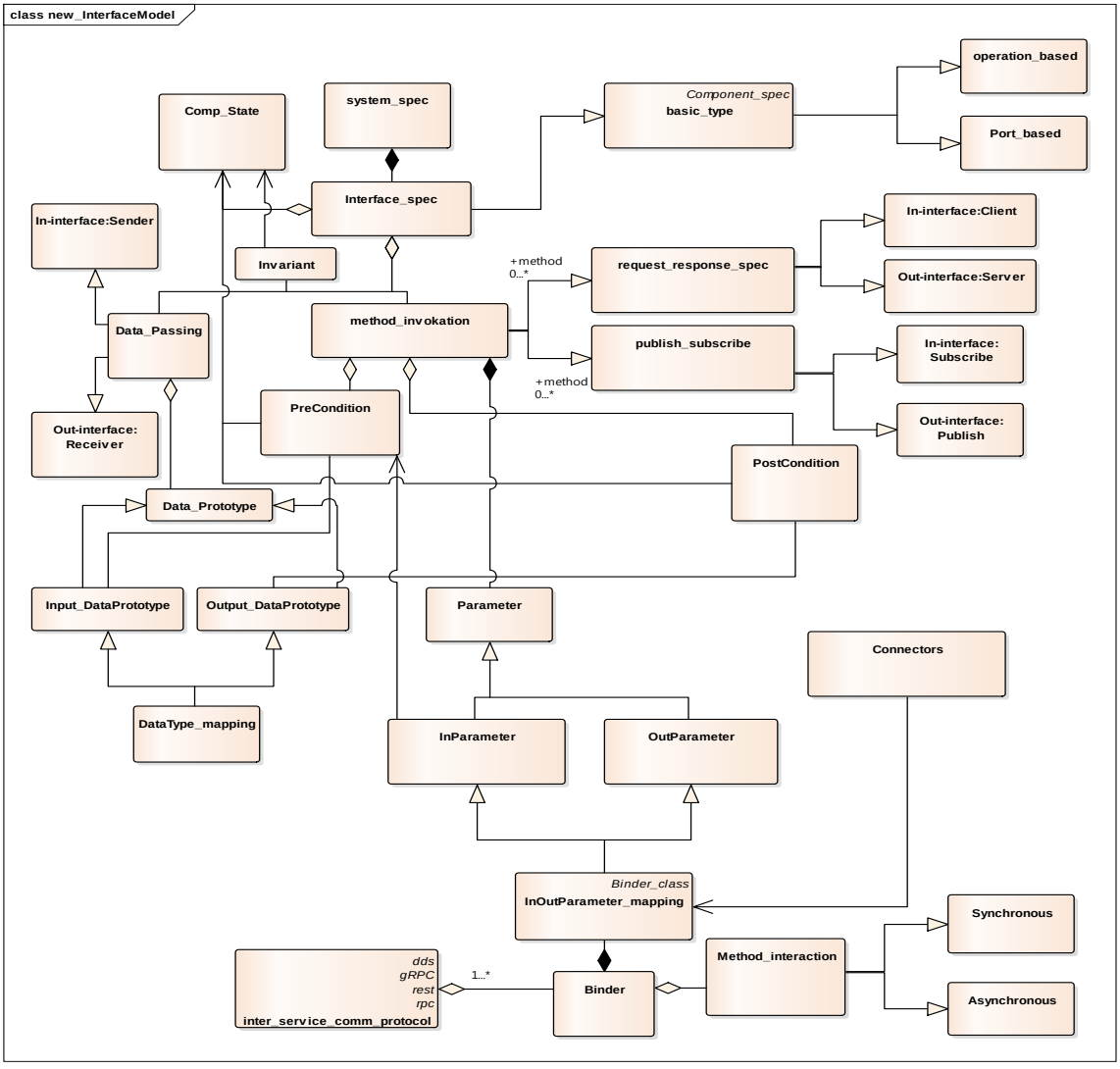

Fig. 2. Abstract hierarchical generic Software Component-Port-Connector (CPC) model based on Black-Box perspective

III. A Semantic Comparison of Component Constructs Between Cross-Domain FWs And Automotive FW

This section provides an overview of abstract SWC interface model descriptions in context of "*constructs*", for those FW components that are used in automotive app domain. The meta-models used for component *constructs* identifies the list of relevant concepts and a list of relevant relationships between these concepts specific to FWs.

A. Automotive Domain: AUTOSAR Adaptive Framework

AUTOSAR (AUTomotive Open System Architecture) is widely accepted as the defacto standard of automotive system software architecture for developing apps of various automotive platforms during the different phases of a vehicle life cycle. The AUTOSAR Adaptive platform (AR AP) app SWC template meta- model is implemented using *ARXML* Schema. The AR AP SWC has a service provider port (PPortPrototype) and a receiver port (RPortPrototype). Each *PortPrototype* is typed using service interfaces. An example of AR AP app SWC (release version 18-10) specific meta-model (M2 level) UML profile representation can be seen in Fig. 3. The Service interface model employed for interface description is specified using various elements, this includes [3]:

- 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 Client-Server Operations in the role of *Methods*. Arguments data required for Client-Server Operation is represented in the role of *ArgumentDataPrototype* in the meta-model as a pre-condition. *Method* calls used for communication among SWC types in AR AP can be described as *synchronous* or *asynchronous* (e.g. *fireAndForget*).

The service interfaces instances in AR AP are deployed using RPC communication.

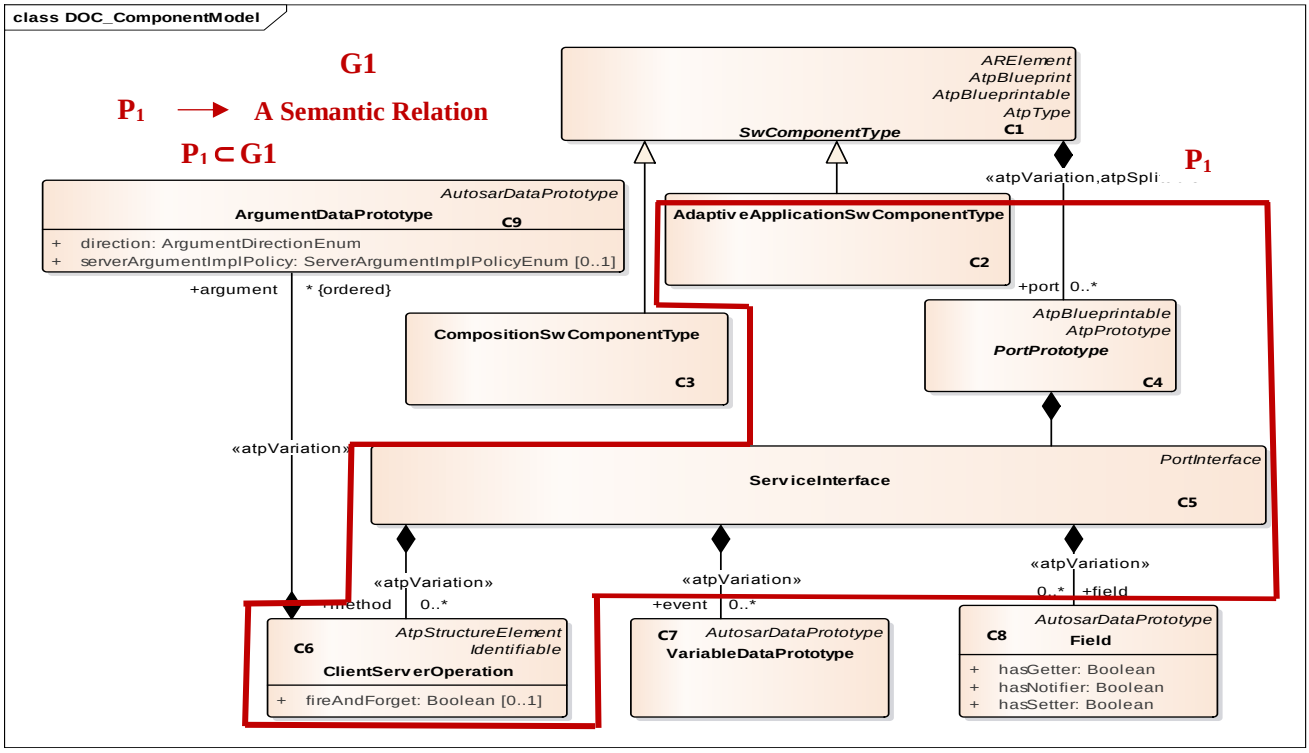

Fig. 3. Overview of Abstract SWC constructs meta-model also named as Graphical Model G1 for AUTOSAR Adaptive Framework

B. Infotainment Domain: Franca Framework

Franca IDL (*FIDL*) is developed as a part of the GENIVI standard Franca (version 0.13.0) FW and supports IVI (In-Vehicle infotainment) system's interfaces. Franca IDL is language binding neutral and independent of concrete bindings. Franca+ IDL (*FCDL*) provides an extension to the Franca FW 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. 4 [10]. Franca FW uses FIDL to define app interfaces and FCDL to define app SWCs and their configurations. Like AR AP, Franca+ FW also supports the *CompositionComponentPrototype (*named as *Component)*. A component contained in a composition is called Component Prototype.

TABLE I. Interface Semantic Synergies of SWC Model: AUTOSAR Adaptive Vs Franca ('C' is *construct* Model element)

AUTOSAR Adaptive Component Construct Element		Franca Component Construct Element
C1	⟸	C10
C3	⟸	C11
C4	⟸	C13
C5	⟸	C14
C6	⟸	C19
C7	⟸	C16
C8	⟸	C17

The *service* attribute marks a component as service running on the target platform. The *Methods*, *Events*, and *Fields* of AR AP service interface are semantically aligned to Franca+ IDL's (*FCDL*) *Methods*, *Broadcasts* and *Attributes*. TABLE I. illustrates interface semantic synergies (indicated by arrows) observed between the app SWC *constructs* (only at app interface level) meta-model elements of AR AP and Franca FWs. Semantic mapping of Franca IDL *Fire-and-Forget Method* to AR AP app service interface *Method* can be achieved by activation of the "*fire & forget*" semantics of a given method by setting the value of attribute *method.fireAndForget* to value true [4].

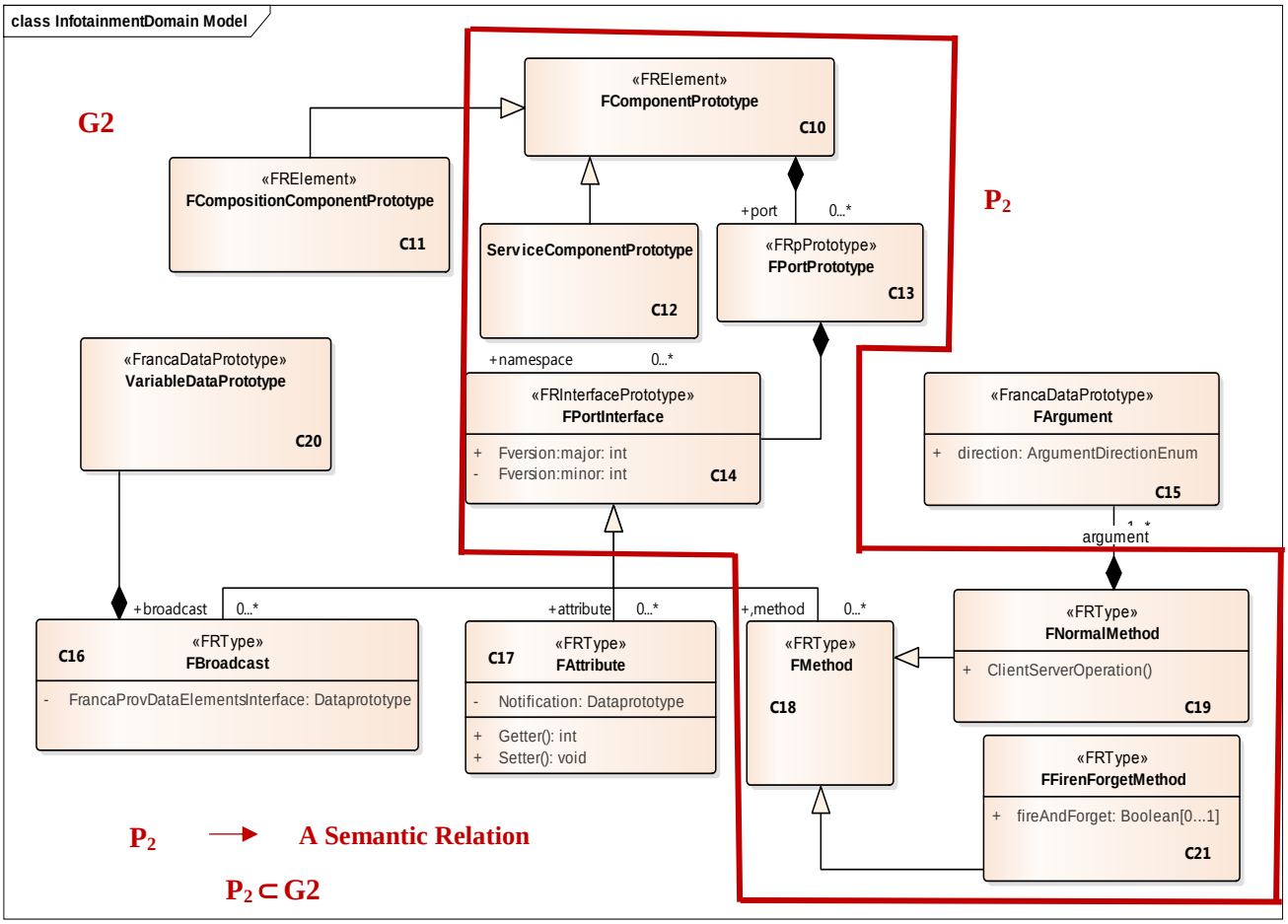

Fig. 4. Overview of Abstract Component Constructs meta-model also named as Graphical Model G2 for Franca (also Franca+) Framework

C. *Robotics Domain: ROS Framework*

The Robot Operating System (ROS) developed by Willow Garage aims to provide a software development environment for robotics. ROS is a perfect FW 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* [8].*Nodes* are semantically similar to *SWComponentPrototype* in AR AP as can be seen in Fig. 5.

A *Topic* can be considered as a named communication channel which is used to send and receive messages between nodes and can be semantically mapped to *PortInterface* of AR AP. In ROS2 (ROS version 2.0) all the necessary information exchange among nodes is performed through *messages*. ROS2 has two basic types of interaction endpoints attached to a *node* that are *data* and *control* interaction endpoints.

In case of the exchanged information having *data semantics* (using DDS: *Data Distribution Services*) and being communicated mostly asynchronously (*non-blocking mode*) as a pre-condition 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* [8]. ROS2 *TopicSubscription* can be semantically mapped to *EventSubscription* in AR AP. *Data Semantics* are semantically similar to the asynchronous *fireAndForget Method* invocation of AR AP [5]. In contrast to AR AP, the concept of a *connection* does not exist in ROS2. Location-transparency between nodes is achieved through the concept of a *master node*. The *master node* provides naming and registration facilities for all nodes [8][5]. In ROS2 in case of the exchanged information having *command semantics* and being communicated mostly synchronously (*blocking mode*) as a pre-condition between invoker and invoke, this functionality is achieved through introduction of a *service* concept. The *service* in ROS2 is defined by a string name and a pair of messages, a *request* and a *reply* message and is semantically similar to RPC based *ClientServerOperation* in AR AP. Unlike AR AP, services cannot be grouped through service ports in ROS2. In ROS2 component models or nodes are described using *Message Description language* (MDL) or *Service Description Language* (SDL) based on *data* and *command semantics* requirements.

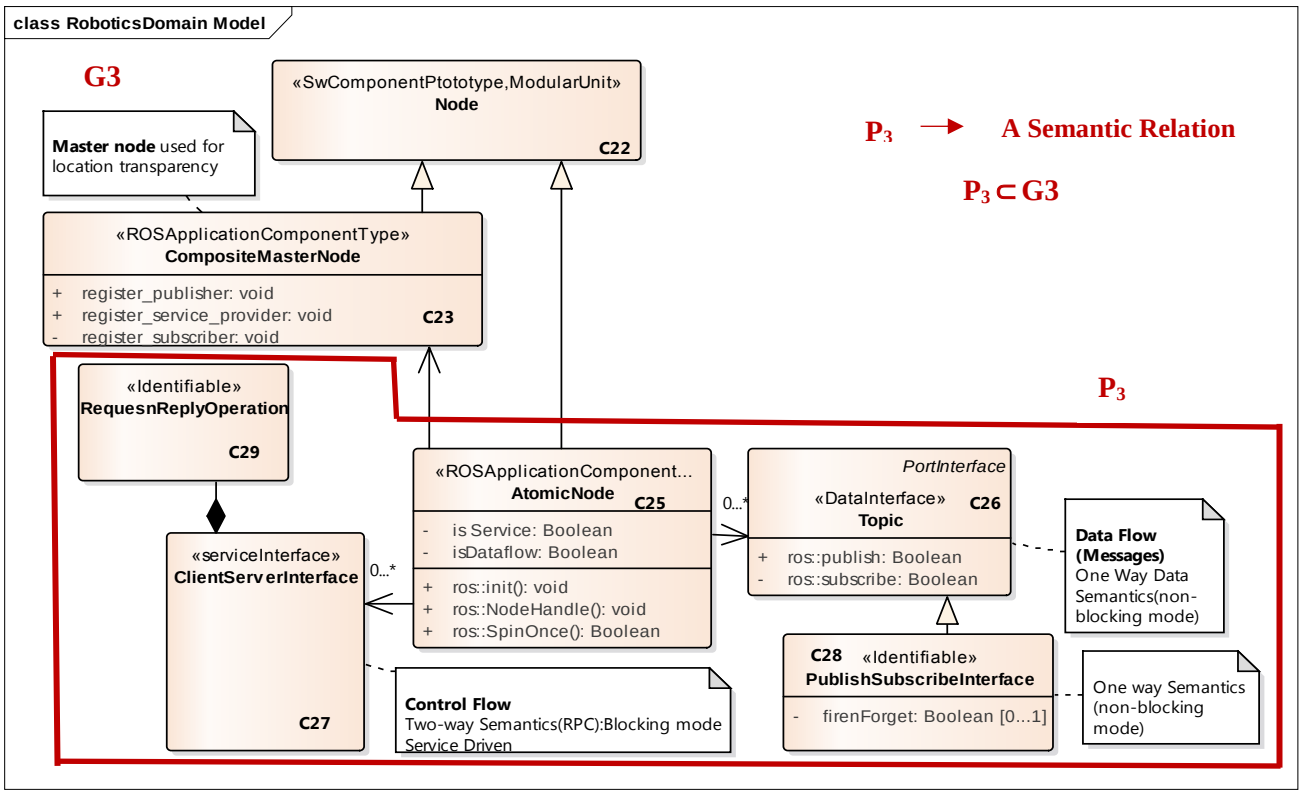

Fig. 5. Overview of Abstract Component Constructs meta-model also named as Graphical Model G3 for ROS Framework

TABLE II. illustrates interface semantic synergies (indicated by arrows) observed between the app SWC construct (only interface) meta-model elements of AR AP and ROS FWs.

TABLE II. INTERFACE SEMANTIC ANALYSIS OF COMPONENT MODEL: AUTOSAR ADAPTIVE VS ROS ('C' IS *CONSTRUCT* MODEL ELEMENT)

AUTOSAR Adaptive Component Construct Element		ROS Component Construct Element
C1	⟸	C22
C2	⟸	C25
C4	⟸	C26
C5	⟸	C27
C6	⟸	C29
C7	⟸	C28

D. Telematics Domain: Android Framework

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. Four different types of app components are used as essential building blocks of an Android app 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* (also an IPC). Intents bind individual components to each other at run-time using messages [13]. However, a data request is treated by the *Content Provider* (CP). The requesting data is indicated through URI (Uniform Resource Identifier), which provides the standard access to CP. The communication between various functionalities of app components is provided by *Receiver of Broadcast and Intents* (RBI). For the communication, the object *Intent* must be passed as a parameter for the RBI to search the functionality. The *broadcast receiver* schedules the *JobServices* event chains. These *Events* are semantically similar to *Events* used by AR AP app SWCs. However, if an app service is used only by the local application and does not need to work across processes, then only a *Binder* class implementation can provide direct client access to public methods in the service [14]. Unlike AR AP apps, for most of the Android apps, the service doesn't need to perform multi-threading, so using a *Messenger* allows the service to handle one call at a time. If it's important that the service to be multi-threaded, use of *AIDL* is preferred to define the interface [13].

The *startService()* service method call invoked by a client result in a corresponding call to the server or service's *Service.onStartCommand* (*Intent*, *int*, *int*) method. On successful service connection binding with the stub or server, the client receives an instance of *IBinder* interface using *onServiceConnected()* callback method as seen in Fig. 6. These method calls of an Android app can be semantically mapped to *ClientServerOperation()* method calls and *Notifier fields* of an AR AP app SWC. The *oneway* keyword modifies the behavior of remote calls. When it is used, a remote call does not block, it simply sends the transaction data and immediately returns. The *oneway* remote method calls can be semantically mapped to asynchronous *ClientServerOperation* or method calls of an AR AP app SWC. If *oneway* is used with a local call, there is no impact and the call is still synchronous.

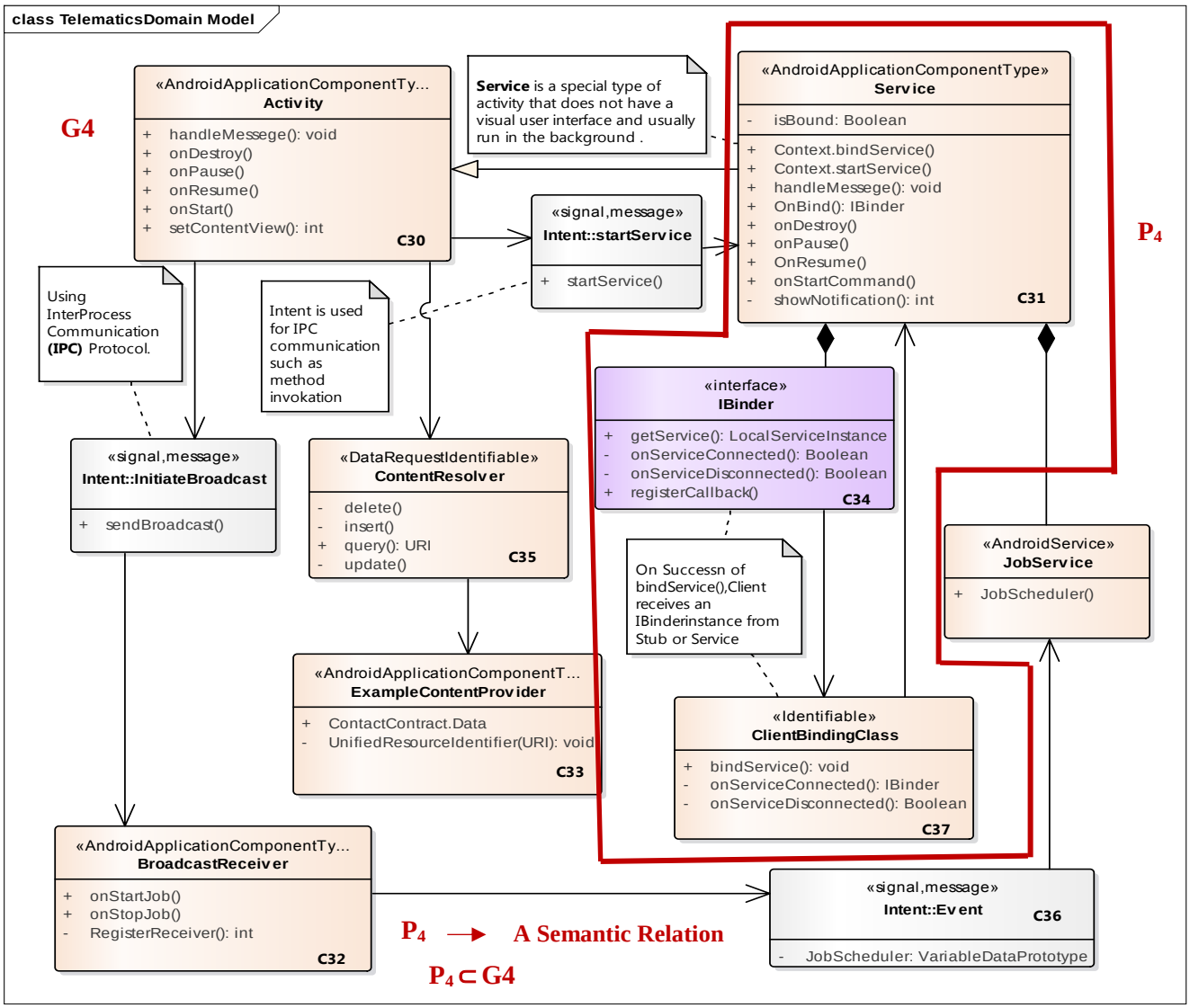

Fig. 6. Overview of Abstract Component Construct meta-model also named as Graphical Model G4 for Android Framework

Unlike AR AP app SWC model, Android app model does not have ports. TABLE III. Illustrates interface semantic synergies (indicated by arrows) observed between the app SWC *construct* (only interface) meta-model elements of AR AP and Android FWs.

TABLE III. INTERFACE SEMANTIC ANALYSIS OF COMPONENT MODEL: AUTOSAR ADAPTIVE VS ROS ('C' IS *CONSTRUCT* MODEL ELEMENT)

AUTOSAR Adaptive Component Construct Elements		Android Component Construct Elements
C1	⟸	C30
C2	⟸	C31
C5	⟸	C34
C6	⟸	C37
C7	⟸	C36

IV. FUTURE SCOPE: SEMANTIC ONTOLOGY MAPPING OF COMPONENT INTERFACE MODELS OF FRAMEWORKS

Aligning semantic ontologies represents a great interest in automotive app domains that manipulate heterogeneous overlapping knowledge FWs. For a future general domain-specific software solution for automotive app interface models, aligning ontologies is a crucial issue in the field of semantic integration, which aims to find semantic correspondences between a pair of elements of ontologies by identifying semantic relations. The interoperability of different UML profile-based component interface models (described in section III) within the same vehicle information system would require a process of detection of interface semantic synergies and resolution of semantic conflicts. The ontologies alignment can use one or more similarity measures (syntactic, semantic and structural) to detect the set of mappings [15]. To better meet this objective, and to significantly increase the scope of future semantic integration algorithms for automotive app interface models following our approach, an overall semantic ontology mapping must be done between the different component construct meta-models.

If we consider G1, G2, G3 and G4 are the graphical model representations of different FW component construct meta-models (as described in section III) and P_1,P_2,P_3 and P_4 are specific semantic relations or ontologies included in G1, G2, G3 and G4 such that $P_1 \subset G1$, $P_2 \subset G2$, $P_3 \subset G3$ and $P_4 \subset G4$. Then based on interface semantic static analysis approach and semantic synergy results, we can say that the semantic ontologies represented by P_1, P_2, P_3 and P_4 are such that $P_1 \cong$

$P_2 \cong P_3 \cong P_4$ with overlapping knowledge domains. Therefore, we can also say that I (G1, G2, G3, G4) is the set of isomorphic or similar sub-graphs [15]. However, such a semantic ontology mapping could be better explained with transformation of the UML profiles in ontologies.

V. Conclusion

Today the development of vehicle software systems is getting more and more complex and widely distributed. End users expect faster, reliable and scalable results despite unpredictable changes in the market. With the proposed approach towards interoperability, we were successful to explore interface semantic synergies among few of the cross-domain component-based software FWs. The app component constructs dimension refers to the notions of reusability and resolvability, which are important principles of CBSE (Component based Software Engineering). The proposed approach of interface static semantic analysis ensures that SWC interface models of heterogeneous frameworks can be compared, correlated and re-used for vehicle apps. The main contribution of this paper is based on the semantic survey of various cross-domain FW SWC interface models from component construct perspective. The FW components considered in the research scope are associated with automotive app domain. Each FW component *construct* is represented in a CPC (Component-Port-Connector) model format using an UML profile (M2 level) representation based on the hierarchically classified three distinct levels: *Semantic*, *Behavior* and *Composition*. The semantic survey of IDLs revealed several areas where they provide common extensive support, both in terms of modeling capabilities and algorithmic (IDL) support. On the other hand, the survey also pointed out areas in which existing IDLs are severely differed from one another.

The static interface semantic analysis approach is at a conceptual stage and is carried out manually. The approach considered the target meta-model as automotive domain SWC *construct* meta-model and the source meta-model as other cross-domain industrial SWC *construct* meta-models, for the semantic mapping (or comparisons). With our approach, we considered AUTOSAR Adaptive app SWC meta-model as target model. The intention of this consideration of the target SWC meta-model is due to the fact that AUTOSAR has been accepted as a de-facto standard for automotive software architecture. The decision for the selection of source and targets meta-models has been made from autonomous driving app's interoperability viewpoint. There is no evolutionary relation between the source and target. In order to transform source models to target models in future or to evolve the model transformation rules from source to target, semantic mappings should be built on the meta-model level and used on the model level.

Considering the context of enterprise interoperability and collaboration that is cross-enterprise, the static interface semantic analysis and comparison approach could simulate the process of integrating the information systems of different enterprises for EIS (Enterprise Integration System) integration. In the last decade, the automotive and other cross-domain research in the field of *Self-driving* has facilitated the development and state-of-the-art so that this technology is evaluated nowadays in large-scales on public roads. In this context of autonomous driving functionality, it is worth to mention that for some of the other existing cross-domain component models that we have already shortlisted, we would like to extend our work in the direction of cross-domain interface semantic analysis and comparison to explore more semantic synergies between these component models and automotive standard FW component models in the future.

References

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

[2] 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.

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

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

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

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

[7] D. Bálek, F. Plášil (2001) "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.

[8] 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.

[9] D.Stampfer, "D2.2.1 State of the Art on Service-Oriented Software Component Models", FIONA ITEA2-12038, version 1.0, March 2014.

[10] 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.

[11] 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.

[12] P.Brada, "A Look at Current Component Models From Black-box Perspective", 35th Euromicro Conference on Software Engineering and Advanced Applications,2009.

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

[14] 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, No. 3 2016, pp. 83-94.

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

[16] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture". The Architecture Journal, 1(1):10-17,2004.

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

[18] I. Malavolta, "Software Architecture Modeling by Reuse, Composition and Customization." Universita di L'Aquila, L'Aquila, Italy, Thesis 1 (2012).

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