Formalizing Correspondence Rules for Automotive Architecture Views

Yanja Dajsuren, Christine M. Gerpheide, Alexander Serebrenik,
Anton Wijs, Bogdan Vasilescu, Mark G. J. van den Brand
Eindhoven University of Technology
5612 AZ Eindhoven, The Netherlands
y.dajsuren@tue.nl, c.m.gerpheide@student.tue.nl, a.serebrenik@tue.nl
{a.j.wijs | b.n.vasilescu | m.g.j.v.d.brand}@tue.nl

ABSTRACT

Architecture views have long been used in software industry to systematically model complex systems by representing them from the perspective of related stakeholder concerns. However, consensus has not been reached for the architecture views between automotive architecture description languages and automotive architecture frameworks. Therefore, this paper presents the automotive architecture views based on an elaborate study of existing automotive architecture description techniques. Furthermore, we propose a method to formalize correspondence rules between architecture views to enforce consistency between architecture views. The approach was implemented in a Java plugin for IBM Rational Rhapsody and evaluated in a case study based on the Adaptive Cruise Control system. The outcome of the evaluation is considered to be a useful approach for formalizing correspondences between different views and a useful tool for automotive architects.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software architectures

Keywords

automotive architecture; architecture framework; architecture view; correspondence rule

1. INTRODUCTION

An architecture framework provides conventions, principles and practices for the description of architectures within a specific domain and/or community of stakeholders [23]. The benefits of existing architecture frameworks such as Kruchten's 4+1 View Model [27], MODAF [32], TOGAF [1], and RM-ODP [22] drive the creation of an architecture framework for the automotive industry, which faces the challenge of tackling increasing complexity and cost of automotive electronic and software components. Currently, an automotive architecture framework [5] and an architecture design framework [19] are being defined with the goal of establishing a standard architecture framework for the automotive industry. However, in these frameworks, the definition of architectural elements including architecture viewpoints, views, and correspondences have not been tackled consistently with automotive Architecture Description Languages (ADLs). Furthermore, the architecture frameworks remain still closed, i.e. difficult to extend with new stakeholder concerns, viewpoints, and views.

1.1 Automotive Architectural Challenges

There are a number of issues which make the development of automotive architecture framework challenging:

- Automotive embedded systems are categorized into vehicle-centric functional domains (including powertrain control, chassis control, and active/passive safety systems) and passenger-centric functional domains (covering multimedia/telematics, body/comfort, and human machine interface (HMI)) [30]. Although each functional domains need to tackle different system concerns (e.g. the power train control enables the vehicle longitudinal propulsion of the vehicle, body domain supports airbag, wiper, lighting etc. for the vehicle users), all the integrated functionalities must not jeopardize the key vehicle requirements, e.g. a safe and efficient way of transporting.
- ADLs like EAST-ADL [7], AADL [18], and AML [3] have been defined for the automotive industry. According to the ISO/IEC/IEEE-42010 or ISO-42010 [23], an ADL provides one or more model kinds (data flow diagrams, class diagrams, state diagrams etc.) as a means to frame some concerns for its stakeholders. An ADL can consist of model kinds, which may be organized into architecture views. Architectural elements e.g. architecture viewpoints, views, and correspondences of the automotive ADLs are not explicitly defined. This results in the loose relation between automotive ADLs and architecture frameworks (which can be improved by refining the definition of the architecture elements of automotive ADLs and architecture frameworks as defined in ISO-42010 standard [23]).
- The automotive industry is vertically organized [4], which facilitates independent development of vehicle parts. An automobile manufacturer (called an "original equipment manufacturer", or OEM) creates the

functional architecture and distributes the development of the functional components to the suppliers, who implement and deliver the software models and/or hardware [4]. (Software models for each functional component or subsystem can be developed in different ADLs or programming languages, which may make the integration process at the OEM more cumbersome.) This process requires common architecture frameworks between OEMs and suppliers or at least better formalization of architecture views and consistency between them.

1.2 Motivation

For the software architecture community, the consistency checking for architecture views and architectural models has been researched vigorously. Specifically, consistency checking for UML diagram types is a well researched area. However, for the automotive architecture field, the issues discussed in the previous section and lack of standard ADL [10] require the consistent definition of architectural elements such as viewpoint, view, correspondence, and correspondence rule. Therefore, we define the automotive architectural elements as part of an Architecture Framework for Automotive Systems (AFAS) and formalize the correspondence between the views extending the expression language.

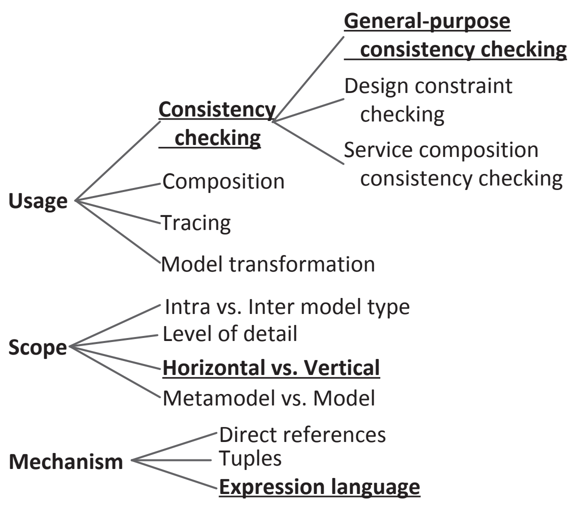


Figure 1: Framework characterizing relations between viewpoints.

In Figure 1, we illustrate a framework, which comprises three dimensions (Usage, Scope, and Mechanism) to characterize relations between views [2]. From four main use cases i.e. consistency checking, composition, tracing, and model transformation, the consistency checking is the focus of this paper. Consistency checking can be used to determine if the information in several views does not conflict with each other [2]. From three example uses of consistency checking relations i.e. general-purpose consistency checking approaches include inconsistency checking of UML diagrams [31]. The scope criteria in Figure 1 defines the range of view relations. From four criteria for the scope, we focus on the horizontal vs. vertical relations. Horizontal relation is used for relations between views at the same abstraction level and the vertical relation is used either relations between views at different abstraction levels or relations between other representations as requirements, detailed design, or implementation [2]. In this paper, our specific scope is in vertical relations i.e. consistency checking between different abstraction levels (e.g. refinement relation). Our approach is based on a language-neutral approach that checks consistency between horizontal and vertical view relations [29]. The mechanism, the third criteria of the framework in Figure 1, categorizes constructs in the architecture description to represent view relations. Our approach extends the existing approach on expression language [25, 29].

1.3 Main Contributions and Outline

This research aims to refine existing automotive architecture frameworks and to formalize the correspondence rules between automotive architecture views. Specifically we focus on the refinement of the structural views, leaving the other views and correspondences for future research. The resulting conformance checking builds upon the hierarchical reflexion model [25] and employs a notion of abstraction similar to that used in the consistency checking approach [29]. As input, the consistency definition requires only a *strength ordering* of the connectors offered by the specific ADL used. In this way the consistency checking approach presented here is applicable to many automotive ADLs. The second contribution of this research is a prototype consistency checking tool in the form of an IBM Rational Rhapsody plugin. Specifically the plugin was developed for SysML [33] structural diagrams.

The remainder of the paper is structured as follows. Section 2 presents the architectural elements of the Architectural Framework for Automotive Systems (AFAS) and architecture views that we have defined based on existing architecture description methods. The scope of the paper is consistency issues between automotive structural views, in particular between functional and software views. These views are presented in the following section. Section 3 describes the ISO-42010 compliant correspondence rules between the functional and software views. Sections 4, 5, and 6 present the consistency semantics, consistency definitions, and tool development, respectively. We evaluate the approach in Section 7 and discuss related work in Section 8. Finally, Section 9 summarizes our contributions and discusses directions for future work.

2. AUTOMOTIVE ARCHITECTURAL FRAMEWORK AND ITS VIEWS

Although automotive architectural frameworks have not been standardized in automotive industry, different types of architecture viewpoints and views as part of automotive architectural frameworks have been introduced recently. The Automotive Architecture Framework (AAF) was defined to describe the entire vehicle system across all functional and engineering domains [5]. This is the first architecture framework for the automotive industry to pave the foundation of a standardized architecture description. The AAF proposes two sets of architectural viewpoints: mandatory or general viewpoints and optional viewpoints. The mandatory viewpoints and their respective views include *Functional viewpoint*, *Technical viewpoint*, *Technical viewpoint*, *Information viewpoint*, *Driver/vehicle operations viewpoint*, and *value net viewpoint*. Optional viewpoints suggested by the

AAF are *safety*, *security*, *quality and RAS* (reliability, availability, serviceability), *energy*, *cost*, *NVH* (noise, vibration, harshness), and *weight* viewpoints. The general viewpoints are intended to be closer to the already proven frameworks in other manufacturing industries e.g. RASDS[1] and RM-ODP[2]. Since the concepts are introduced in the first draft of the AAF, there needs to be further research to identify automotive specific architectural elements.

An Architectural Design Framework (ADF) is defined to support the construction of an architecture framework for the automotive industry [19]. The ADF supports only the system design process ("Technical processes") of the ISO/IEC 15288 standard[3] and derived from the SAGACE method [19]. The ADF includes *operational*, *functional*, *constructional*, and *requirements* viewpoints. Although the AAF and ADF are defined to provide the basis for the architecture framework for the automotive industry, architectural viewpoints and views are extracted from architecture frameworks from other industries. Therefore, we defined an Architectural Framework for Automotive Systems (AFAS) as illustrated in Figure 2 based on the study of proprietary automotive architectural models and practices, draft automotive architecture frameworks as AAF and ADF, and automotive ADLs like EAST-ADL [7], AADL [18], AML [3], and TADL [38]. The AFAS framework thus contains architectural viewpoints and views complementary to automotive ADLs.

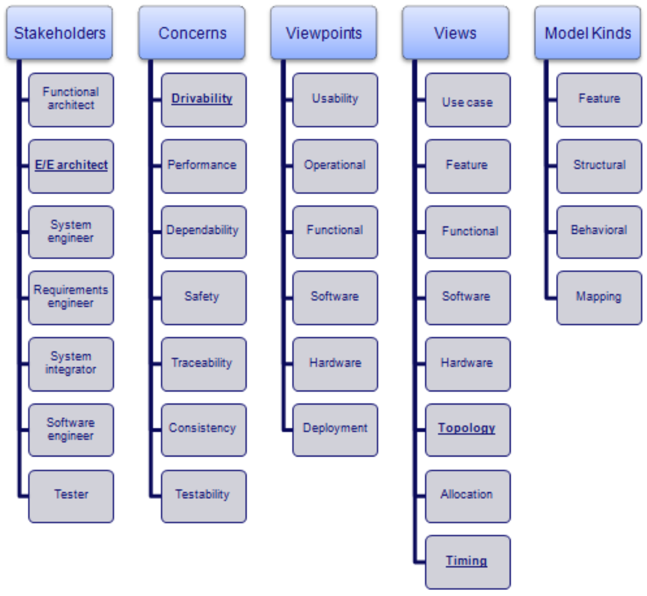


Figure 2: AFAS overview.

- *Use case view* shows the interaction between users and the system.
- *Feature view* captures the vehicle product line features, such as cruise control or bluetooth telephone connection, which can be configured for a product or a specific vehicle.
- *Functional view* specifies a structural model that contains a number of functions or subsystems realizing features.
- *Software view* represents the software architecture, where detailed descriptions and implementation of a function is realized in software components or blocks.
- *Hardware view* represents the electrical/electronical (E/E) hardware architecture. The hardware architecture typically consists of electronic control units (ECUs), sensors, actuators and Controller Area Network (CAN) busses.
- *Topology view* specifies the connections (buses e.g. CAN, Local Interconnect Network (LIN) and wires etc.,) between ECUs, sensors, and actuators.
- *Allocation view* describes the mapping between software components to ECUs.
- *Timing view* specifies timing analysis such as bus schedulability analysis and CPU response time analysis views of the system.

In Figure 2, the architectural elements of the AFAS are similar to other architectural framework elements. Only E/E architect, Driveability, Topology and Timing views are specific for automotive domain. Although Functional and Software views exist in other frameworks, the correspondences between these views are specific to the automotive domain. As mentioned in the Introduction, functional decomposition is carried out by the OEMs and the functional models are delivered to the supplier, who elaborates the model in the software view and delivers back the functionality in the ECU. The elaboration of the functionality in software view may take several iterations e.g. the feedback to the functional model of the OEM or changes in the functional model need to be propagated into the supplier's software models. This process is currently rather document-centric and error-prone [19]. Therefore, improving consistency between these views is a step towards a semantic consistency of architectural modeling between OEMs and suppliers.

3. ARCHITECTURE CORRESPONDENCE

Consistency between views is one of the key problems in functional and software architectures [17]. Although software consistency has been a focus of the research community for many years, it has not been fully addressed in the automotive architecture views. Since more than a decade automotive ADLs have been developed and automotive architecture frameworks have been created recently. However, there is no standard ADL and architecture framework for the automotive industry yet. The architecture views and correspondences between them are not explicitly defined, which hinders semantic consistency of architectural modeling between OEMs and suppliers. Therefore, in this section we define the notion of correspondence and correspondence rules between functional and software views with the purpose of expressing and enforcing consistency among these views. Although OEMs and suppliers may use different ADLs, for illustrating the correspondences, functional and software models are represented in SysML structural diagrams.

In the ISO-42010 standard [23], a *correspondence* defines a relation between architecture description (AD) elements,

[1]Reference Architecture for Space Data Systems: `http://public.ccsds.org/publications/`
[2] Reference Model of Open Distributed Processing: `http://www.rm-odp.net`
[3]ISO/IEC 15288: http://www.15288.com/

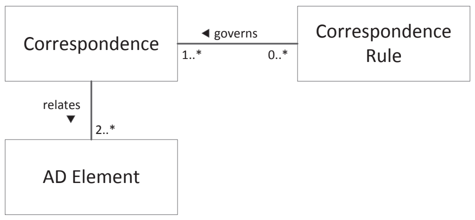


Figure 3: AD elements and correspondences [23].

which in the context of this paper is called the architecture view. Architecture relations can be, e.g. *refinement*, *composition*, *consistency*, and *traceability* [23]. Correspondences can be governed by *correspondence rules* as depicted in Figure 3. We specify below the correspondences, which define a refinement relation between functional and software views. Although the ISO-42010 standard does not specify a format for correspondences, they can be defined as relations and tables [16]. The following correspondence and correspondence rule are defined based on the refinement relation between architecture views (functional and software view) as illustrated in Figure 4.

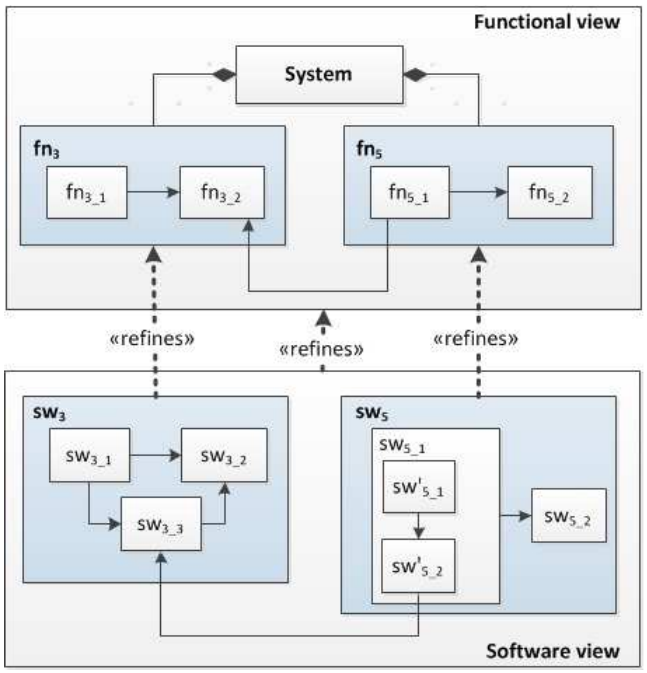


Figure 4: Example of functional and software views.

Consider functional and software views of a system S as a functional view, *FN(S)*, and a software view, *SW(S)*. Given that *FN(S)* includes functional components, $fn_1, \ldots, fn_n$ and *SW(S)* has software components, $sw_1, \ldots, sw_r$, a correspondence expressing which functional components are refined by which software components is specified in Table 1.

The correspondence rule for the refinement correspondence between functional and software views is:

R1: Every functional component, *fn*, defined by the functional view *FN(S)*, needs to be refined by one or more software components, *sw*, as defined by the software view *SW(S)* of a system S.

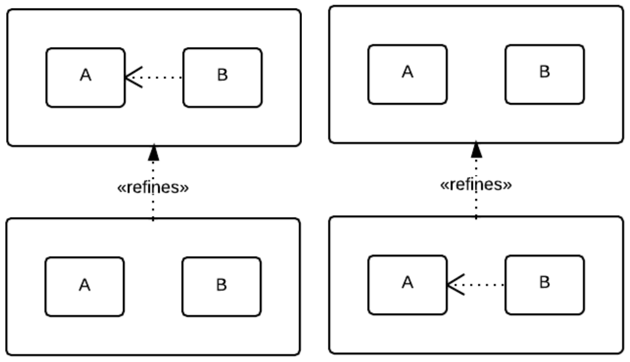
Table 1 Refinement correspondence.

SW(S) refines FN(S) See rule: R1	
sw_3	fn_3
sw_5	fn_5

In Figure 4 and Table 1, we have for example sw_3 *refines* fn_3 , and sw_5 *refines* fn_5.

4. CONSISTENCY SEMANTICS

Checking the consistency between the functional view and the software view involves checking the refinement correspondence between a high-level model and a low-level model. To perform this check, it suffices to perform a model transformation of the refined model into a model with the same level of abstraction as the high-level model. In the hierarchical reflexion model [25], this transformation is referred to as a *lifting* operation. Lifting abstracts from the details inserted into the refinement, leaving only information relevant for comparison with the high-level model. In terms of static models, this requires that for every entity present in the high-level model, the relationship present in the low-level model must be derived. The high-level model can then be directly compared with the lifted model. Possible inconsistencies are relations which exist in the high-level model but not in the lifted model, or relations which exist in the lifted model but not in the high-level model. These inconsistencies are referred to as *absences* and *divergences*, respectively, as depicted in Figure 5.

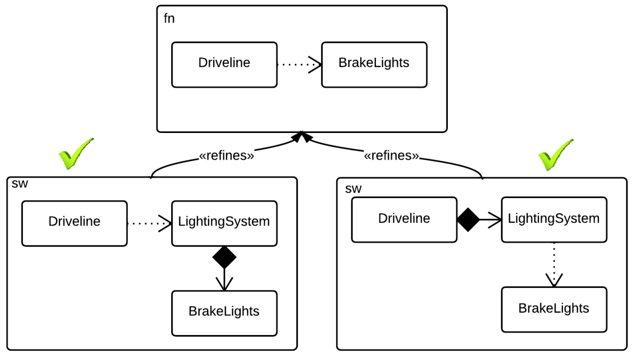


(a) Absence **(b)** Divergence

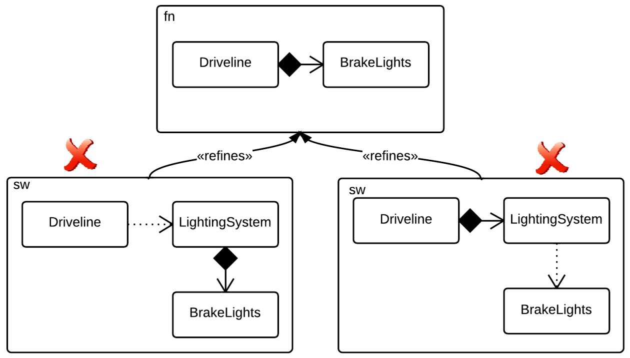
Figure 5: Absence and divergence inconsistencies

To perform the lifting operation, we first generalize the hierarchical reflexion model. This generalization is necessary because the ADLs used in the automotive industry are more expressive than what can be accommodated by the reflexion model itself, where only *partof* (e.g. composition) and *reference* (e.g. dependency) relation types are considered. Specifically, rather than distinguishing relations, we impose a strength ordering on ADL connectors. This not only generalizes the vocabulary of the model, but also allows an extended role of transitivity in the model to more connector types than that used in the reflexion model, therefore substantially increasing the expressiveness of the consistency definition.

To illustrate the role of connector strength, Figure 6 depicts two similar functional models in SysML, each with two possible software refinements. Figure 6a shows a functional model with a dependency relation. In the left-hand refinement of Figure 6a, a wrapper entity was inserted. Semantically this still indicates that `Driveline` makes an (indirect) call to `BrakeLights`. Therefore the derived relationship between `Driveline` and `BrakeLights` is a dependency relationship, and should be considered consistent with the functional model. In the right-hand refinement, the entity `DriveLine` was refined to specify that in fact a child entity `LightingSystem` makes a call to `BrakeLights`. In this case clearly the derived relationship is again dependency. Therefore, regardless of whether the dependency relationship preceded, the derived relationship when combined with composition was still semantically a dependency. This suggests that connectors can in fact be arranged in an *ordering* according to their relative strengths. The lifting operation then involves transitively applying the ordering to allow stronger connectors to override weaker ones.



(a) Consistent dependency refinements



(b) Inconsistent composition refinements

Figure 6: Semantic differences between dependency and composition refinements

In Figure 6b, the same refinements are now refining a composition relation in the functional model. However, an automotive architect would not consider either proposed refinement to be consistent, since splitting up a composite entity into two entities which communicate via function calls was not intended by the architect. Therefore, applying the connector ordering (which again derives only implicit dependencies in the refinements), again correctly yields an inconsistency. It is essential however to note that the high-level model must be taken into account when performing the lifting operation on the low level model. The original hierarchical reflexion model, due to its use of the full transitive closure in composition, simply extracts *all* implicit relationships between all elements in the low-level model, and then performs a comparison with the high-level model. However, this approach would yield many false positives due to the loss of information incurred during the transformation. This problem is illustrated in Figure 7. There, if a full transitive closure is taken to extract all implicit (lifted) dependencies, the two low-level models cannot be distinguished. However, it is clear that the left-hand refinement should be consistent with the functional model, while the right-hand refinement should not, in that case because the refinement is violating the layering specified by the high-level model. Therefore, it is essential that the lifting operation does not use a full transitive closure to derive the lifted relationships from the refinement. Instead, the strength ordering should only be applied transitively until an entity also present in the high-level model is encountered.

5. CONSISTENCY DEFINITION

In this section, we formalize the inconsistency checking approach explained in the previous section.

Let $REL = \{rel_1, \ldots, rel_k\}$ be an ordered set of relations available in the automotive ADL, where rel_1 is the weakest relation and rel_k is the strongest one. For the SysML examples provided in Section 4, $REL = \{composition, dependency\}$.

Next we introduce two new operations, which we term the *right-lifting* and *left-lifting* operations on the low-level model (i.e. software model), denoted as $\overrightarrow{R_i}$ and $\overleftarrow{R_i}$ in Equations 1 and 2, respectively. These relations are defined for each connector $rel_i \in REL$. Below, FN represents the functional model, SW is the software model, $rel_i(A,B)$ is a direct relation of type rel_i between entities A and B in FN, and $\widetilde{rel_i}(A,B)$ is a direct relation of type rel_i in SW. The lifting operation is carried out on the software model.

$$\overrightarrow{R_i}(X,Y) \quad \Leftrightarrow \quad \bigvee_{j \leq i} \widetilde{rel_j}(X,Y) \wedge Y \notin FN \qquad (1)$$

$$\overleftarrow{R_i}(X,Y) \quad \Leftrightarrow \quad \bigvee_{j \leq i} \widetilde{rel_j}(X,Y) \wedge X \notin FN \qquad (2)$$

Note that $\overrightarrow{R_i}^{+}(X,Y)$ and $\overleftarrow{R_i}^{+}(X,Y)$ are the irreflexive transitive closures of $\overrightarrow{R_i}(X,Y)$ and $\overleftarrow{R_i}(X,Y)$, respectively. The right-lifting operation represents a series of elements in the software model connected by relations, where (at least) the target entities of each relation exist only in the software model, and connectors are equal or weaker in strength to rel_i. The same intuition holds for the left-lifting operation, except the restriction there is that at least all relation *sources* only exist in SW. As illustrated in examples from Figure 6 and 7, incorporating the entities present in the high-level model while performing the lifting operation clearly distinguishes our method from previous research utilizing hierarchical reflexion.

In the Equation 3, we redefine the *lifted relation* between two model entities, denoted $\widetilde{rel_i}^{\uparrow}$, incorporating the left- and right-lifting operations. This new definition takes into

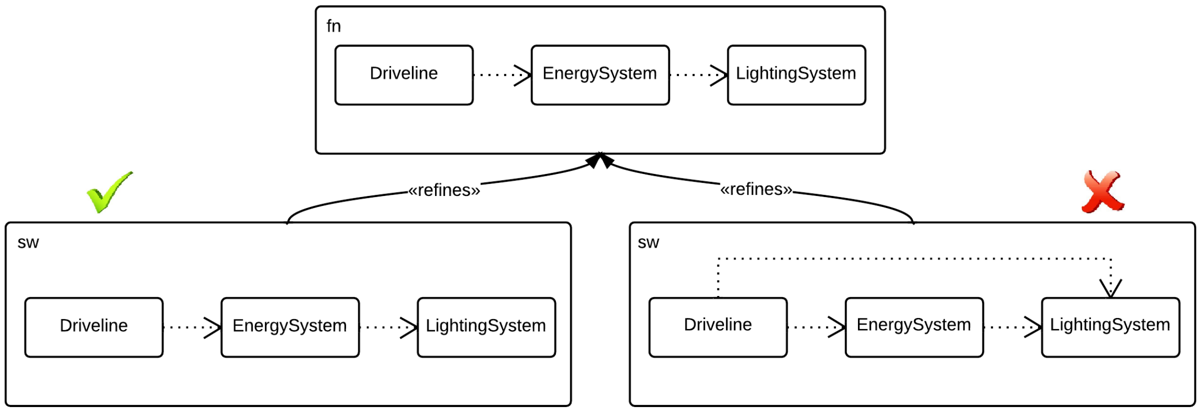


Figure 7: Inadequacy of using full transitive closure to extract relations in refinements

account the cases where either two entities are directly connected by relation rel_i, or there exist one or more intermediary entities in between which appear only in SW and are connected by relations which are equal or weaker than rel_i. A and B are functional and software model entities.

$$\widetilde{rel_i}^{\uparrow}(A,B) \Leftrightarrow \quad \widetilde{rel_i}(A,B) \vee \exists C,D \in SW \backslash FN : \Big(\overrightarrow{R_i}^{+}(A,C) \wedge \widetilde{rel_i}^{*}(C,D) \wedge \overleftarrow{R_i}^{+}(D,B)\Big) \tag{3}$$

where $\widetilde{rel_i}^{*}$ is the reflexive transitive closure of $\widetilde{rel_i}$.

The lifting operation should then be performed on SW using all entity pairs A,B that appear in FN, resulting in a lifted low-level model. The lifted model can then be straightforwardly compared to the high-level model to check for absent and divergent relations for each relation type, presented in Equations 4 and 5.

$$absence_{rel_i}(A,B) \Leftrightarrow rel_i^{+}(A,B) \wedge \neg\widetilde{rel_i}^{\uparrow}(A,B) \tag{4}$$

$$divergence_{rel_i}(A,B) \Leftrightarrow \neg rel_i^{+}(A,B) \wedge \widetilde{rel_i}^{\uparrow}(A,B) \tag{5}$$

6. TOOL DEVELOPMENT

In the following sub-sections, the algorithm for checking inconsistencies based on the definitions in the previous section, the details for the tool implementation, and a description of using the tool are presented.

6.1 Checking Algorithm

The lifted model that is calculated by applying the Equations 4 and 5 results in a low-level model, e.g. software model, that has been abstracted from all details not already present in the high-level model, e.g. functional model. Comparing this lifted model to the high-level model then fulfills the intuition of consistency described initially by Dijkman et al [12]. Furthermore, calculating a lifted model and then applying consistency checks for *absence* and *divergence*, rather than working directly on the low-level model, has been seen to significantly improve the scalability and maintainability of consistency checking algorithms in practice [14].

The implementation of the consistency checking algorithm is described below. Note that in addition to the *absence* and *divergence* checks, an additional check is run to make sure that all blocks from the high-level model exist in the low-level model; if not, an *absentBlock* error is reported.

```
Algorithm CheckConsistency(FN, SW)
Input: FN is the high-level functional model, and SW is
    the low-level software model
Output: A(possibly empty) set of consistency errors
1.  Encode FN and SW as directed graphs, where edges
    are annotated with the relation type (dependency or
    composition)
2.  Let $\tilde{SW}$ be a new, empty graph to contain the lifted
    model of SW
3.
(∗ Populate the lifted model of SW ∗)
4.  for all elements $A, B \in FN$
5.      do
6.          if $\widetilde{rel_{dep}}^{\uparrow}(A,B)$ is true
7.              then Add an edge from A to B to $\tilde{SW}$ anno-
                    tated with dependency
8.          if $\widetilde{rel_{comp}}^{\uparrow}(A,B)$ is true
9.              then Add an edge from A to B to $\tilde{SW}$ anno-
                    tated with composition
10.
(∗ Check for absence ∗)
11. for each edge $e = (A, B) \in FN$
12.     do
13.         if $A \notin \tilde{SW}$ (or $B \notin \tilde{SW}$)
14.             then Report error absentBlock(A) (or
15.                     absentBlock(B), respectively)
16.         if e is annotated with dependency $\wedge$
17.                 $absence_{dep}(A,B)$ is true
18.             then Report error absentDependency(A, B)
19.         if e is annotated with composition $\wedge$
20.                 $absence_{comp}(A,B)$ is true
21.             then Report error absentComposition(A, B)
22.
(∗ Check for divergence ∗)
23. for each edge $e = (A, B) \in \tilde{SW}$
24.     do
25.         if e is annotated with dependency $\wedge$
26.                 $divergence_{dep}(A,B)$ is true
27.             then Report error
28.                     divergentDependency(A, B)
29.         if e is annotated with composition $\wedge$
```

```
30.                 $divergence_{comp}(A, B)$ is **true**
31.         **then** Report error
32.                 $divergentComposition(A, B)$
```

While there are many parts of the algorithm which could be optimized for a faster running time, they are left out here to improve readability. Furthermore, although the algorithm accepts only two models as input, it can be used to check deeper models (i.e. where a high-level model is refined by another model, which in turn refined by yet another model), simply by running the *CheckConsistency* algorithm for each pair of parent-child models.

6.2 Tool Implementation

A prototype tool was implemented as a Java plugin integrated into the IBM Rational Rhapsody for SysML structural diagrams, i.e., Block Definition Diagram (BDD) and Internal Block Diagram (IBD). The reason for this choice is three-fold: First, IBM Rational Rhapsody is a well-established, enterprise modeling tool for designing complex software products including automotive software systems [21]. In addition to support for SysML, Rational Rhapsody also supports UML and some domain-specific languages (DSLs). So, a plugin developed for use with SysML is easily convertible to a tool for other supported languages. Second, it is important for the tool to be integrated directly into the development environment. This not only increases usability by allowing architects to work with a tool they already understand, but also increases the likelihood that consistency checks are run often. This integration is possible in Rational Rhapsody because it offers a comprehensive Java API for plugin development. Finally, IBM Rational Rhapsody is well-documented and has an active developer community, making it a low-risk choice for development.

In addition to the Rational Rhapsody API functions, the Java Universal Network/Graph (JUNG) library was used for encoding the graphs required to represent the high-level, low-level, and lifted low-level models. Using a third-party, comprehensive graph library greatly reduced the complexity required to implement the checking algorithm. As output, the tool notifies the user of all absences and divergences encountered inside the error pane.

6.3 Using the Tool

The consistency checking plugin expects a project to have at least two SysML Package elements: a high-level package and a low-level package. A top-level element of Package was chosen to separate the high-level from the lower-level models because it was seen that some automotive companies already organize their models this way.

Each Package should contain exactly one SysML BDD, which describes the Blocks relevant for that Package together with their dependency and composition relationships. Then there should be another diagram, which we term Overview, which specifies which Packages refine which other Packages. A refinement is specified by adding a Dependency relation between the Packages in the Overview diagram with the «refine» stereotype.

The consistency check tool can run on the Overview diagram by selecting the *Tools > Check Model* command. The plugin will then find all refinements described in the Overview diagram. For each refinement relation, it will retrieve the high-level and low-level models (in this case Block Definition Diagrams) and run the plugin's *CheckConsistency* algorithm. When errors are found, a new bottom frame opens with a list of all the errors, noting exactly which diagram, relation, and specific offending elements caused the consistency error, according to the list of failed elements generated during plugin execution. After performing this check, the architect can resolve the errors or alert another architect that there are errors, and then rerun the check.

7. EVALUATION

In this section we evaluate the tool implemented for the inconsistency checking as presented in Section 6. We applied the tool to an Adaptive Cruise Control (ACC) system. ACC, Figure 8, is a cruise control system with enhanced functionality assisting the driver to keep a safe distance to other traffic ahead and alerting her if manual intervention is required [8]. In our evaluation of the prototype inconsistency checking tool, two student teams emulated an OEM and a supplier. The students follow a Master of Science in Automotive Technology, therefore they have experience in modeling automotive systems. The "OEM" team created a functional architecture for a truck and handed in a functional model of the ACC (Figure 9a) to the "supplier" team. The "supplier" team elaborated the ACC software model (Figure 9b) and created a running ACC prototype. In the real life automotive modeling case, at this phase the supplier software would be integrated the ACC by the OEM and tested thoroughly.

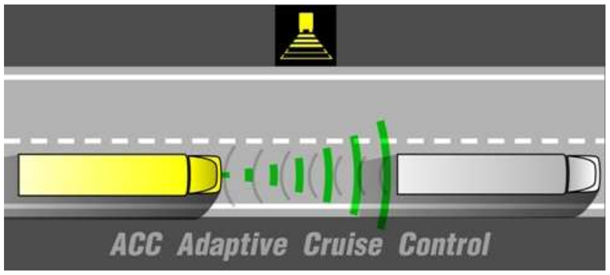


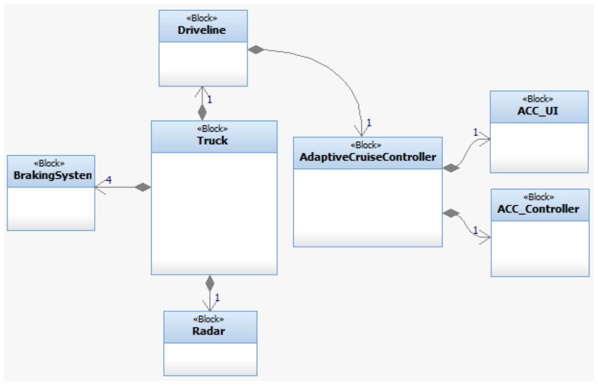
Figure 8: Adaptive cruise control [8].

Although ACC subsystem works correctly according to the OEM specification, the ACC software model created by the "supplier" team is inconsistent with the functional model provided by the "OEM" team. Indeed, using the prototype inconsistency checking tool with *Tools > Check Model*, the divergence relations between ACC_Controller and ACC_UI, and Driveline and Radar are detected. These relations are missing in the functional view shown in Figure 9a.

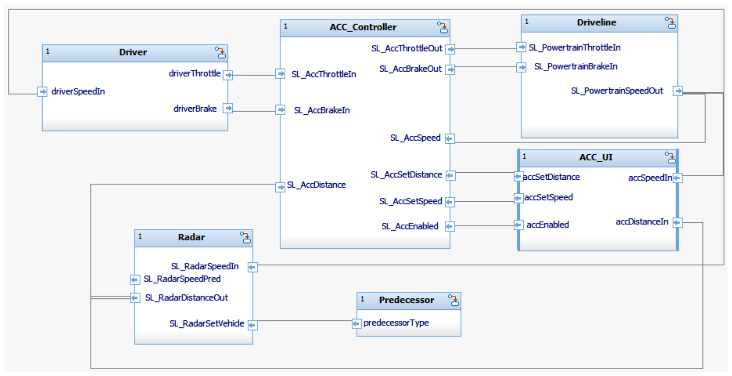
Early inconsistency detection by the prototype tool was considered useful by both teams. The team members appreciated that the consistency checks are executed only when specifically invoked by the architect. This is in sharp contrast with a recommendation of Rosik, Buckley and Ali Babar [36] that argue that consistency errors should be reported continuously during development.

8. RELATED WORK

MEGAF (MEGamodeling Architecture Frameworks) infrastructure is developed to enable reusable architecture frameworks [20]. Due to the limitation of the MEGAF tool support for extracting the architectural elements from the



(a) ACC functional view



(b) ACC software view

Figure 9: Consistency checking between functional and software views of the ACC system

automotive ADLs and architecture frameworks, we defined the AFAS manually by analyzing state-of-the-art architecture description mechanisms. Our approach is based on the hierarchical reflexion model [25, 29]. Specifying correspondences between viewpoints has been introduced in [35] [34]. OCL is used for implementing the correspondence rules in this approach. However, in this approach, views are expressed as UML models which are not widely used in the automotive architectural modeling. Our approach extends this approach by enabling a technique to specify intentional correspondences for automotive architecture modeling.

In their overview of UML consistency management, Elaasar and Briand [15] describe viewpoint unification to transform one UML view to another. Since different UML diagram types contain different sorts of information, this process often resulted in lost information and furthermore only appropriate for certain diagram pairs. Such transformation-based consistency approaches are employed by many authors [28] [6] [37]. However, the desire of the researchers to keep the operation generic for many domains and diagram types results in only basic consistency rules. For example, a rule may guarantee that classes with a certain name exists. Since we consider only the refinement correspondence, more powerful rules can be formulated.

In the UML Analyzer tool [13], Egyed presents a rule-based approach to abstract from entities and relations which exist in a refinement model, resulting in a scalable consistency checking tool [14]. Furthermore it was found to be beneficial for both performance and usability to separate the transformation (abstraction) phase from the consistency checking phase. However, the rules are limited to UML, making them not directly applicable to automotive ADLs. Furthermore, the rule format does not lend itself to generalization, in contrast to a generic mathematical definition for consistency. Some authors choose to translate architectural diagrams to an intermediate language, for example XMI [39] [26], to take advantage of the existing power for expressing consistency rules available in those languages. Such representations are however considerably less intuitive, whereas using a graph representation can already maintain the structure and information present in most automotive ADLs while requiring a less radical model transformation.

In the hierarchical reflexion model [25], relations that exist in a parent model are checked to exist in a *lifted* model which has been derived from source code. This approach is useful since it can equally be applied to checking two hierarchical models against each other. It is also highly intuitive and results in few false positives [24]. Previous work in automotive ADL consistency has adapted the reflexion model to the automotive domain [10]. There, multiple levels of automotive models are considered. Furthermore, the research presented here extends [10] by providing more sound consistency rules for the functional and software views.

9. CONCLUSION AND FUTURE WORK

In this paper, first we presented architecture description elements including architecture views of the AFAS framework based on the study of state-of-the-art architecture description mechanisms for automotive systems. To further revise the framework, we plan to use the MEGAF approach [20], since the MEGAF is built upon the ISO/IEC/IEEE-42010 standard and enables the definition of a reusable and open architecture framework. Although consistency issues between architecture viewpoints and views were tackled before in the software industry, there is still a need to develop a method to enforce the consistency between different views of OEMs and suppliers.

Therefore, in this paper we also proposed an inconsistency detection approach based on correspondence rules between automotive architecture views. We focus on the refinement relationship between functional and software views, where the functional models are refined by adding more details in the software view. The revised definition for consistency proposed here requires only that an ordering be imposed on the connector types available in a given ADL, allowing it easily to be used with many automotive ADLs. Then a prototype tool was developed for IBM Rational Rhapsody which can perform this consistency checking between different architecture views. The inconsistency checking approach and the prototype tool were evaluated in the scope of an Adaptive Cruise Control modeling among two separate teams emulating an OEM and automotive supplier. The early inconsistency detection by the prototype tool was considered useful by both teams.

The future work will be improving the prototype tool after carrying out a comprehensive case study in an industrial setting by extending the consistency rules and overall usability of the tool. The end result is expected to be a powerful consistency checking approach and useful tool for automotive system and software architectures. The preliminary demonstration of the prototype has already attracted the attention of potential users and received some insightful feedback. Support for consistency checking between the other automotive views identified in Section 2 is also planned. This will be integrated into the automotive-specific quality framework that comprises quality specification, measurement, and evaluation methods targeting both architectural and design models [11] [9].

10. ACKNOWLEDGEMENT

Yanja Dajsuren, Alexander Serebrenik and Mark G.J. van den Brand are supported by the Dutch Automotive Innovation Programme (HTAS) project (HTASI10002). Bogdan Vasilescu gratefully acknowledges support from the Dutch Science Foundation (NWO) through the project NWO 600. 065.120.10N235.

11. REFERENCES

[1] The Open Group Architectural Framework (TOGAF). `http://www.opengroup.org/togaf/`.

[2] N. Boucké, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh. Characterizing relations between architectural views. In R. Morrison, D. Balasubramaniam, and K. Falkner, editors, *Software Architecture*, volume 5292 of *Lecture Notes in Computer Science*, pages 66–81. Springer Berlin Heidelberg, 2008.

[3] P. Braun and M. Rappl. A model-based approach for automotive software development. In P. Hofmann and A. Schürr, editors, *OMER*, volume 5 of *LNI*, pages 100–105. GI, 2001.

[4] M. Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 33–42. ACM, 2006.

[5] M. Broy, M. Gleirscher, S. Merenda, D. Wild, P. Kluge, and W. Krenzer. Toward a holistic and standardized automotive architecture description. *Computer*, 42(12):98–101, 2009.

[6] R. Buhr. Use case maps as architectural entities for complex systems. *Software Engineering, IEEE Transactions on*, 24(12):1131–1155, 1998.

[7] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M. Reiser, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren, and M. Weber. The EAST-ADL Architecture Description Language for Automotive Embedded Software. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 297–307. Springer Verlag, 2011.

[8] DAF Trucks N.V. Adaptive Cruise Control. `http://www.daf.com/SiteCollectionDocuments/Products/Safety_and_comfort_systems/DAF-ACC-EN.pdf`, 2013.

[9] Y. Dajsuren, A. Serebrenik, R. Huisman, and M. G. J. van den Brand. A quality framework for evaluating automotive architecture. In *Proceedings of the FISITA World Automotive Congress*, 2014.

[10] Y. Dajsuren, M. G. J. van den Brand, A. Serebrenik, and R. Huisman. Automotive ADLs: a study on enforcing consistency through multiple architectural levels. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures*, pages 71–80. ACM, 2012.

[11] Y. Dajsuren, M. G. J. van den Brand, A. Serebrenik, and S. Roubtsov. Simulink models are also software: Modularity assessment. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 99–106. ACM, 2013.

[12] R. Dijkman, D. Quartel, and M. van Sinderen. Consistency in multi-viewpoint design of enterprise information systems. *Information and Software Technology*, 50(7):737–752, 2008.

[13] A. Egyed. Automatically validating model consistency during refinement. In *23rd International Conference on Software Engineering (ICSE 2001), Toronto, Ontario, Canada*, pages 12–19, 2000.

[14] A. Egyed. Scalable consistency checking between diagrams - The VIEWINTEGRA approach. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 387–390. IEEE, 2001.

[15] M. Elaasar and L. Briand. An overview of UML consistency management. *Carleton University, Canada, Technical Report SCE-04-18*, 2004.

[16] D. Emery and R. Hilliard. Every architecture description needs a framework: Expressing architecture frameworks using ISO/IEC 42010. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 31 –40, 2009.

[17] G. Fairbanks and D. Garlan. *Just Enough Software Architecture: A Risk-Driven Approach.* Marshall & Brainerd, 2010.

[18] P. Feiler, D. Gluch, and J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, 2006.

[19] H. Góngora, T. Gaudré, and S. Tucci-Piergiovanni. Towards an architectural design framework for automotive systems development. In M. Aiguier, Y. Caseau, D. Krob, and A. Rauzy, editors, *Complex Systems Design and Management*, pages 241–258. Springer Berlin Heidelberg, 2013.

[20] R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione. On the composition and reuse of viewpoints across architecture frameworks. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, WICSA-ECSA '12, pages 131–140, Washington, DC, USA, 2012. IEEE Computer Society.

[21] IBM. Rational Rhapsody Designer for systems engineers. `http://www.ibm.com/software/products/`.

[22] ISO. ISO/IEC 10746-1 Information technology – Open Distributed Processing – Reference Model: Overview. December 1998.

[23] ISO/IEC/IEEE 42010:2011. Systems and software engineering—architecture description. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=50508`, 2011.

[24] J. Knodel and D. Popescu. A comparison of static architecture compliance checking approaches. In *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*, pages 12–12. IEEE, 2007.

[25] R. Koschke and D. Simon. Hierarchical reflexion models. In *Proceedings of the 10th Working Conference on Reverse Engineering*, page 36. IEEE Computer Society, 2003.

[26] Y. Kotb and T. Katayama. Consistency checking of UML model diagrams using the xml semantics approach. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 982–983. ACM, 2005.

[27] P. B. Kruchten. The 4+1 View Model of architecture. *Software, IEEE*, 12(6):42–50, Nov. 1995.

[28] D. Liu, K. Subramaniam, B. Far, and A. Eberlein. Automating transition from use-cases to class model. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 2, pages 831–834. IEEE, 2003.

[29] J. Muskens, R. Bril, and M. R. V. Chaudron. Generalizing consistency checking between software views. In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pages 169–180, 2005.

[30] N. Navet and F. Simonot-Lion. *Automotive Embedded Systems Handbook*. CRC Press, Inc., USA, 2009.

[31] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelsteiin. xlinkit: A consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)*, 2(2):151–185, 2002.

[32] B. M. of Defence. MOD Architecture Framework. `http://www.modaf.org.uk/`.

[33] OMG. Systems Modeling Language (SysML) Specification version 1.2. `http://www.sysml.org/specs`, 2010.

[34] J. Romero, J. Jaen, and A. Vallecillo. Realizing correspondences in multi-viewpoint specifications. In *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International*, pages 163–172, 2009.

[35] J. Romero and A. Vallecillo. Well-formed rules for viewpoint correspondences specification. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, pages 441–443, 2008.

[36] J. Rosik, J. Buckley, and M. Ali Babar. Design requirements for an architecture consistency tool. In *21st Annual Psychology of Programming Interest Group Conference*, pages 1–15, 2009.

[37] B. Shishkov, Z. Xie, K. Lui, and J. Dietz. Using norm analysis to derive use case from business processes. In *5th Workshop on Organizations semiotics. June*, pages 14–15, 2002.

[38] The TIMMO Consortium. TADL: Timing Augmented Description Language version 2. `http://www.timmo-2-use.org/timmo/index.htm`.

[39] C. Zapata, G. González, and A. Gelbukh. A rule-based system for assessing consistency between UML models. *MICAI 2007: Advances in Artificial Intelligence*, pages 215–224, 2007.