Software and Systems Modeling (2023) 22:891–917
https://doi.org/10.1007/s10270-022-01022-z

REGULAR PAPER

Reference architectures modelling and compliance checking

Alessio Bucaioni[1] · Amleto Di Salle[2] · Ludovico Iovino[3] · Ivano Malavolta[4] · Patrizio Pelliccione[3]

Received: 30 August 2021 / Revised: 27 June 2022 / Accepted: 6 July 2022 / Published online: 6 August 2022

Abstract
Reference architectures (RAs) are successfully used to represent families of concrete software architectures in several domains such as automotive, banking, and the Internet of Things. RAs inspire architects when designing concrete architectures, and they help to guarantee compliance with architectural decisions, regulatory requirements, as well as architectural qualities. Despite their importance, reference architectures still suffer from a number of open technical issues, including (i) the lack of a common interpretation, a precise notation for their representation and documentation, and (ii) the lack of conformance mechanisms for checking the compliance of concrete architectures to their related reference architecture, architectural decisions, regulatory requirements, etc. This paper addresses these two issues by introducing a model-driven approach that leverages (i) a domain-independent metamodel for the representation of reference architectures and (ii) the combination of model transformation and weaving techniques for the automatic conformance checking of concrete architectures. We evaluate the applicability, effectiveness, and generalizability of our approach using illustrative examples from the web browsers and automotive domains, including an assessment from an independent practitioner.

Keywords Model-driven Engineering · Software architecture · Reference architecture

1 Introduction

From the internet of things [1] to automotive [2], embedded systems [3], and web services [4], today many industrial sectors are heavily based on software. Due to their increasing complexity, heterogeneity, and dynamic nature, the design and operation of those software systems are becoming challenging. Evidence shows that one of the most critical success factors for the design and development of complex software systems is raising the level of abstraction by focusing on their *software architectures* [5–7]. According to Bass et al., a software architecture of a system can be defined as the set of its "software elements, relations among them, and properties of both" [5].

A Software Reference Architecture (RA) is a general architecture that is used as a foundation for the design of concrete architectures[1] within a given context or application domain, e.g. automotive [8]. The role of RAs is to aggregate knowledge, abstract solutions, and design expertise within a targeted domain [8]. Software architects can use RAs as an inspiration for the design of concrete architectures in multiple contexts or as a standardization tool within the targeted application domain [5,9]. Martínez-Fernández calculated "a three-year ROI of 42% with a payback period of 16,5 months and 7 applications" for software companies adopting RAs [10].

Responsible Editor: Dimitris Kolovos.

✉ Ludovico Iovino
ludovico.iovino@gssi.it

Alessio Bucaioni
alessio.bucaioni@mdh.se

Amleto Di Salle
amleto.disalle@univaq.it

Ivano Malavolta
i.malavolta@vu.nl

Patrizio Pelliccione
patrizio.pelliccione@gssi.it

[1] Mälardalen University, Västerås, Sweden

[2] University of L'Aquila, L'Aquila, Italy

[3] Gran Sasso Science Institute, L'Aquila, Italy

[4] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

[1] In the remainder of this paper we will use (i) the term *reference architecture* for referring to a software reference architecture and (ii) the term *concrete architecture* for indicating an instance of a software reference architecture.

Besides being used as templates for designing concrete architectures, a RA also has the important role of guaranteeing compliance with architectural decisions, regulatory requirements, as well as architectural qualities such as modularity, reusability, flexibility, and usability. As a concrete example, we can mention AUTOSAR, a mature RA that is widely adopted in the automotive domain [11]. AUTOSAR aims to standardize the ECU (Electronic Control Units) software architecture and facilitate interoperability among components. Many original equipment manufacturers (OEMs) adopt it, and suppliers (both tier 1 and tier 2)[2]. Referring to other domains, *e.g.* cloud computing, Yimam et al. [12] observe that "building compliant and secure cloud systems have many challenges because of the complex nature of regulations and cloud systems" and they highlight that this compliance might be reached via RAs. As another example in this field, we can mention the Compliance Reference Architecture Framework (RAF) of VMware. RAF provides a consistent way to assess and evaluate the compliance with key industry and government regulations of cloud environments[3].

In the last years, RAs have been successfully used across many industrial sectors such as automotive [13], avionics [14], and robotics [15]. Despite the relevance of RAs, several challenges are still plaguing their wide adoption [8]. To date, there is no consensus on how to precisely represent RAs [8] and on how to check and/or guarantee the compliance of an architecture to the identified reference architecture. As a result, both practitioners and researchers tend to represent RAs only informally [16]. In their empirical study, Martínez-Fernández et al. put the "lack of common interpretation" as one of the first seven drawbacks of using RAs [17]. Concrete architectures are designed by software architects, who spend a considerable amount of time trying to ensure the compliance of concrete architectures to their corresponding RA [9]. Such a task entails a kind of typing purely based on methodology and domain knowledge, which is rarely captured by current Architectural Languages (AL) [18][4]. As a consequence, software architects have no accurate instruments for ensuring compliance of concrete architectures to corresponding RAs, and they risk making wrong assumptions about the system and its context of use, poor design decisions, and sub-optimal design solutions [8,9,19]. Engineers from a multinational software consulting company interviewed by Martínez-Fernández et al. reported "inefficient support for adaptation and instantiation" as a top drawback related to RAs [17]. When prompted about what they would change with respect to RAs, they chose a list of improvements, including technological change and practices or guidelines to RAs [17]. They stated that they would like a "visual plugin to facilitate the development of components and automate the job" [17]. Moreover, existing approaches [20,21] highlight the need for consistency/conformance checking between the implemented architecture and the intended one. These approaches deal with the consistency of the implemented system with the intended architectural model, leaving out the consistency checking of the intended architecture with the chosen reference architecture.

In this paper, we contribute toward defining a way to model reference architectures, guarantee their compliance with guidelines and rules through suitable conformance checks, and check the conformance of concrete architectures to their target reference architecture. We make use of Model-Driven Engineering (MDE) [22] and its techniques and technologies since, as also testified by previous works [16,23], they are interesting instruments to be utilized for RAs. Specifically, we propose a model-driven approach, named MORE, for (i) modelling reference architectures, (ii) automatically checking the conformance of RAs to architectural styles, and (iii) automatically performing conformance checks of concrete architectures against reference architectures. The main building blocks of the proposed approach are:

- A domain-independent metamodel for representing RAs;
- A validation script for checking the well-formedness of a modelled RA with respect to architectural styles;
- A model transformation for promoting RA models into their corresponding domain-specific metamodels;
- A domain-independent weaving metamodel for suitably mapping the entities of concrete architecture models to the entities of a RA;
- A validation script for the automatic compliance checking of concrete to reference architectures.

We use examples from the web browsers and automotive domains to evaluate the proposed approach's applicability, effectiveness, and generalizability. In particular, we exploit the work of Grosskurth and Godfrey on web browsers [24] for demonstrating the *applicability* of our approach. Then, starting from the RA for web browsers, we use a model mutator [25] for generating faulty architectures and demonstrating the *effectiveness* of the proposed approach in identifying compliance issues between concrete and reference architectures. Finally, we leverage the works in [26] and in AWS Archi-

[2] In July 2021, it involves 305 partners (9 core, 1 strategy, 60 premium, 60 development, 148 associate, and 27 attendees) https://www.autosar.org/.

[3] VMware's Approach to Compliance: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/solutions/vmware-approach-to-compliance.pdf; Industry support for the VMware Compliance Reference Architecture: https://news.vmware.com/releases/pci-compliance-supporting-partner-quotes-10412.

[4] In the context of this study we use the acronym AL (architectural language) instead of ADL—as done in [18]—to highlight that we refer to any form of expression used for architecture description.

tecture Center [5] from the automotive domain for showcasing the *generalizability* of the proposed approach.

The *target audience* of this paper includes standardization bodies (*e.g.* OMG[6]), technology working groups (*e.g.* those belonging to the W3C consortium[7]), software architects, and MDE researchers. Standardization bodies and technology working groups can use our approach for modelling and reasoning on reference architectures within their domains. For example, the Web Standards Project[8] can use MORE for modelling concrete architectures of web browsers and automatically checking the conformance to a pre-existing reference architecture, similarly to what we do in Sect. 5. Software architects can use the proposed approach for modelling concrete architectures of their systems and automatically checking their compliance to (standardized) RAs with little to no additional effort. Researchers and practitioners in MDE might find the proposed approach inspiring for investigating new techniques, concepts, and other relationships among architectural artefacts.

Structure of the paper The remainder of this paper is organized as follows. Section 2 provides background for reference architecture. Sections 3 and 4 present the proposed approach and its core components together with the usage of MDE. Section 5 describes the evaluation of the proposed approach. Section 6 discusses strengths and limitations of the proposed approach, while Sect. 7 reports the related work. Section 8 closes the paper with final remarks and future work.

2 Reference architecture

Various definitions of RAs have been proposed in the last years [5,27,28]. According to the Rational Unified Process (RUP) [28], a RA is a predefined set of architectural patterns, partially or entirely instantiated, proven for use within specific business and technical contexts, together with artefacts enabling their use. Bass et al. [5] define a RA as a model composed of software elements and data flows among them.

A vast body of literature is devoted to defining RAs for different domains. In the automotive domain, the authors in [29] propose a functional (system functions and connections among them) RA for autonomous vehicles. The work in [30] extends this RA for autonomous vehicles and presents a functional reference architecture for cars as constituents of a System of Systems (SoS). A further example of a functional RA in the automotive domain can be found in [31], where the authors present a horizontal and vertical layered architecture for dealing with new automotive trends. The work in [32] describes three technical reference architectures representing three generations of automotive systems. In the web browser domain, the work in [24] describes a RA for web browsers based on two open-source implementations. In this work, we will use the RA in [24] as the application scenario. In the cloud domain, the work in [33] presents a RA of the SeaClouds solution, which aims at enabling a seamless adaptive multi-cloud management of complex applications.

Several works have been using the concept of RA for defining so-called Architecture Frameworks (AFs) [34]. AFs provide means for documenting, understanding, modelling, analysing, using and comparing RAs. AFs can be organized into different categories (layers) depending on their level of generality: Meta Architecture Frameworks (MAFs), Common Architecture Frameworks (CAFs), Domain-specific Architecture Frameworks (DAFs), Company-specific Architecture Frameworks, Product Line Architecture Frameworks, and Product Architecture Frameworks [35]. The works in [35] and [36] propose a CAF used for designing a DAF for the automotive domain. Other examples of the domain and company-specific architecture frameworks can be found in [2,37–39]. In particular, the authors in [2] propose an AF for Volvo Cars, while the authors in [37–39] propose different AFs for the automotive domain.

As highlighted in [40], further research in evaluating the economic viability of adopting RAs is strongly needed. Also, we are witnessing a shortage of models to precisely evaluate the benefits of RAs [41]; this is needed in practice to make informed decisions about their adoption. An interesting contribution is provided by the work in [17], which investigates the benefits and drawbacks perceived by various stakeholders in nine RAs designed by a multinational software consulting company (further details might be found in Sect. 7). Also, according to Gartner's report, the benefits of RAs include (i) reducing the complexity of hardware and software architecture, (ii) increasing speed, (iii) reducing operational expenses, and (iv) quality improvements [42]. This report also highlights that organizations pay for the lack of architecture and configuration standards in terms of higher costs and less agility.

Martínez Fernandez [10] conducted an empirical investigation on the benefits and drawbacks of using software reference architectures in the industry. Their study focuses on AUTOSAR as the reference architecture for developing automotive software systems. The main identified benefits are standardization (88% of respondents), reuse (80%), interoperability (51%), improved communication (47%), and reduced costs (39%). The study identified complexity (65% of respondents) as the main drawback of AUTOSAR, also stating that complexity grows proportionally with the size of the project. Practitioners suggested that tool support is the main solution for addressing complexity in their AUTOSAR-based projects. Other drawbacks identified by practitioners

[5] https://aws.amazon.com/architecture/.

[6] https://omg.org.

[7] https://www.w3.org.

[8] https://www.webstandards.org.

include significant initial investment (59%), steep learning curve (51%), confusion about terminology (41%), too high-level of abstraction (35%), inefficient instantiation (22%), and poor documentation (20%). Our proposed approach can support practitioners in addressing several drawbacks mentioned above, especially concerning complexity, steep learning curve, and inefficient instantiation.

3 MORE: modelling reference architectures and automatic compliance checking

In this section, we conceptually describe MORE, an approach that enables architects to automatically check the compliance of their architecture description to modelled reference architectures. In Sect. 4, we automate MORE through the use of MDE. We present MORE and its automation through MDE separately to emphasize that MDE is just one of the alternatives that architects and engineers may use for automating the proposed approach. We also contribute with a language to describe reference architectures, as better detailed in Sect. 4. Within MORE, we identify the following *stakeholders* and related responsibilities:

- Domain expert The domain expert is a practitioner with a proven and extensive experience within a specific domain, possibly matured within several companies. The domain expert knows the nature of software systems in that domain and she is responsible for designing RAs (MyRA in Fig. 1).
- Senior architect The senior architect has experience in designing and developing architectures and complex software systems. Senior architects can be either practitioners or established researchers in the domain of software architectures and they are responsible for defining the rules describing the architectural styles used.
- Architect The architect is a practitioner working on a specific project within a given domain. She is responsible for creating concrete architectures (MyA in Fig. 1) and for ensuring the conformance of concrete to reference architectures.
- Approach maintainer The approach maintainer is either a practitioner or researcher. She is responsible for creating and maintaining the artefacts of the proposed approach. This stakeholder will be visible in Sect. 4, where we describe the implementation of MORE through the use of MDE technologies.

As shown in Fig. 1, MyRA is a RA, which is modelled by a domain expert according to the Generic Reference Architecture grammar (GRA), which is domain independent and encodes constraints, rules, and characteristics of the RA. The domain expert is needed to define a domain specific RA specification. Indeed, the support of a software architect would be needed in order to properly formulate the RA. A RA is typically modelled once, stored in a repository of RA descriptions, and used when there is the need to instantiate it into an architecture description. Section 4 describes GRA and its implementation using MDE technologies. Similarly, an architecture description, called MyA, is expressed according to the Architecture Language Grammar (ALG), which encodes constraints, rules, and characteristics of the considered architecture language (AL). Similarly to what done in [18], with AL we refer to any formal or informal language that permits to produce an architecture description. To concretely model an architecture description a software architect might use one of the available ALs surveyed in [18].

Once a RA is specified, i.e. MyRA, together with an architecture specification, i.e. MyA, MORE can run various checks in order to verify various types of conformance.

- *GRA conformance* (Conf_{GRA}): MyRA should conform to constraints, rules, and characteristics of the reference architecture. This is checked by the function:

  $$\text{Conf}_{\text{GRA}} : \text{GRA} \times \text{MyRA} \rightarrow \{0, 1\}$$

- *ALG conformance* (Conf_{ALG}): MyA should conform to constraints, rules, and characteristics of the ALG. This is checked by the function:

  $$\text{Conf}_{\text{ALG}} : \text{ALG} \times \text{MyA} \rightarrow \{0, 1\}$$

- *Architectural style conformance* (Conf_{AS}): guidelines from architectural style can be formulated as a set of rules: $ASG = \{r_1, ..., r_n\}$. Different styles can be formulated using different set of rules. The conformance to the architectural styles might be encoded in a function returning $true$ when the reference architecture MyRA complies to the architectural styles specification ASG, $false$ otherwise:

  $$\text{Conf}_{\text{AS}}\text{ASG} \times \text{MyRA} \rightarrow \{0, 1\}$$

- *MyRA conformance* ($\text{Conf}_{\text{MyRA}}$): the architectural elements in the architecture specification MyA should respect the constraints defined on top of the elements defined in MyRA:

  $$\text{Conf}_{\text{MyRA}} : \text{MyA} \times \text{MyRA} \rightarrow \{0, 1\}$$

The conformance functions that we described enable the automatic checks that permit to assess whether an architecture specification is compliant to a reference architecture.

Without automation mechanisms, all the above checks must be performed manually by the architects or the engi-

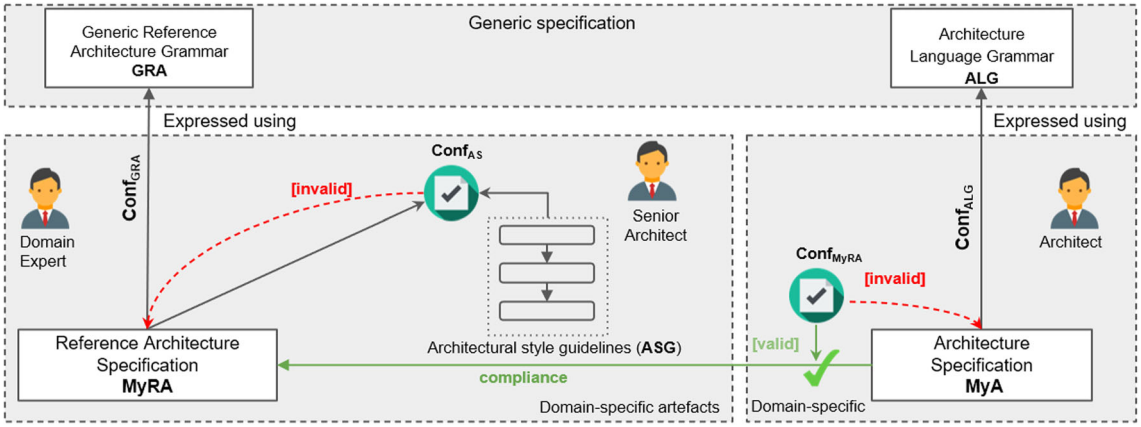


Fig. 1 Overview of the MORE approach

neers. At times, this is not only tedious and impractical but infeasible. RAs may consist of tens of software elements and connections and comply with more than one pattern. Besides, software architectures may conform to more than once RA. In this context, it is evident that, even when feasible, manual interventions are not desirable as they are prone to errors. In the following section, we describe how the concepts described in this section can be automated through the use of MDE.

4 Implementing MORE using MDE

In this section, we describe how MDE can be used to automate the conceptual description of MORE provided in Sect. 3. Specifically, we describe how MORE supports the formal representation of RAs and the automatic conformance checking of concrete to reference architectures. We define the concept of *virtual conformance* inheriting the concept of *conformance* typical of MDE [43]: in the same way a model conforms to a metamodel, a concrete architecture *virtually* conforms to a reference architecture [16]. We use the term *virtual* for highlighting that this new kind of conformance is not bound to a strict typing relation, as in the case of a conformance relation between a model and its metamodel, rather it formalizes an indirect *compliance* relationship [44].

Figure 2 builds on Fig. 1 and provides an overview of the proposed approach in terms of its main components and involved stakeholders. In particular, the proposed approach includes the following *components*:

- *Generic reference architecture metamodel (GRAmm)* GRAmm is a metamodel for designing arbitrary RAs (My Reference Architecture model - MyRA in Fig. 2) in terms of components and connectors. Reference architectures (MyRA in Fig. 2) are modelled via a dedicated graphical editor based on the GRAmm. Within the proposed approach, GRAmm can be used for describing RAs from different domains.
- *Architectural style validation script (ASV)* As RAs are typically designed according to architectural styles, ASV is a script for validating the conformance of a given RA (MyRA in Fig. 2) to a (set of) specified architectural style(s). For instance, ASV can check whether a given RA conforms to the layered architectural style. The proposed approach leverages different ASVs for different architectural styles of interest. However, ASVs can be used within different domains.
- *Model promotion transformation (T)* T is a model transformation, which transforms a (model of a) RA (MyRA in Fig. 2) into a domain metamodel (My RA Metamodel - RAmm in Fig. 2) as a crucial step towards the conformance checking of concrete to reference architectures. Within the proposed approach, T can be used to transform any model of RAs.
- *AL metamodel (ALmm)* ALmm is a metamodel for designing concrete architectures (My Architecture Model - MyA in Fig. 2). As for GRAmm, ALmm is generic and can be used within several domains.
- *RATagging weaving (RAW)* RAW is a metamodel for linking elements of concrete architectures (MyA) to elements of a RA (RAmm) via a *typeOf* relationship. The interplay of RATagging Weaving (RAW), Generated RA Validation Script (RAV) and the RAmm (which is the result of the model promotion transformation (T)) implements the check $\text{Conf}_{\text{MyRA}}$ of Fig. 1.
- *RA validation script (RAV)* RAV is a script for validating the conformance of concrete architectures (MyA in Fig. 2) to RAs (MyRA in Fig. 2).

With respect to the goals of the proposed approach, GRAmm and ASV provide the means for the formal representation of RAs, while T, ALmm, RAW, and RAV are in charge of checking the virtual conformance.

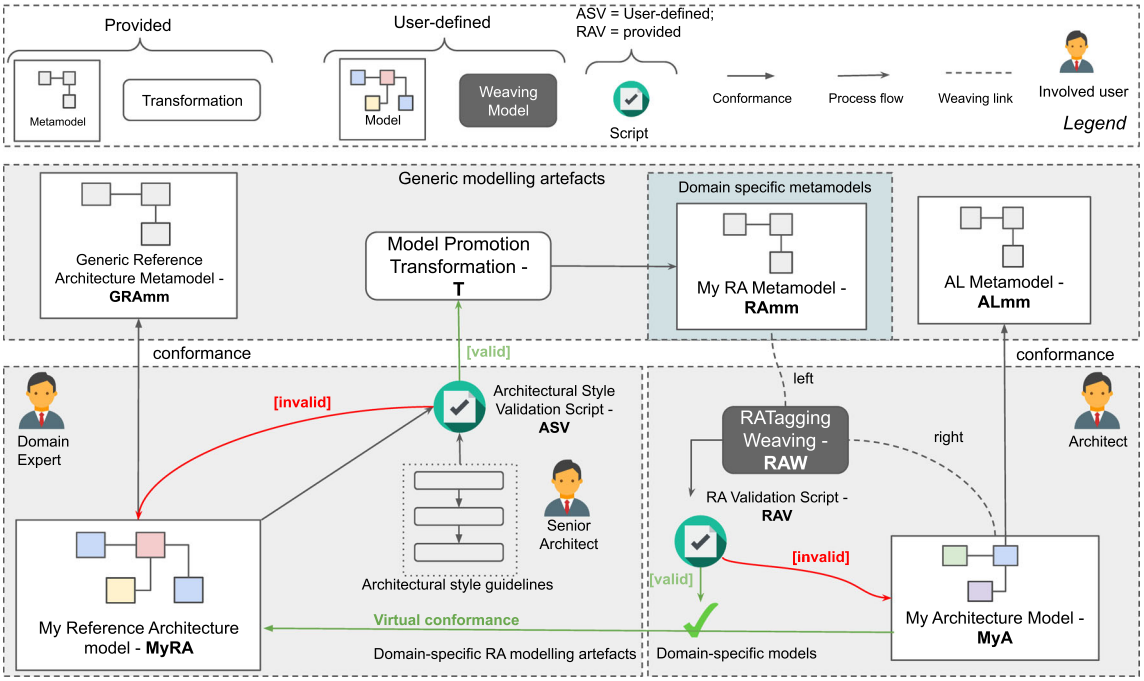


Fig. 2 Implementing MORE using MDE

In the following we present the six steps of a typical usage scenario of the proposed approach.

- *First step* The domain expert creates a model of a RA (*i.e.* MyRA in Fig. 2). This is done by using the generic metamodel GRAmm defined by the approach maintainer. The modelled RA is visualized in a dedicated graphical editor, in order to show the declared components and connectors.
- *Second step* The senior architect defines the validation script ASV for checking the compliance of MyRA to the architectural styles. In fact, elements (components and connectors) of a RA are typically composed according to a predefined (set of) architectural style(s) [45].
- *Third step* If MyRA does not comply with the specified architectural styles (red dotted line in the left part of Fig. 2), ASV provides the senior architect with a list of violations. Moreover, the visualization of the reference architecture is updated with the validation errors in order to guide the resolution. If MyRA complies to the specified architectural styles (solid green arrow in the left part of Fig. 2), then T can be invoked on MyRA. T uses the principle of *promotion* [46] in the field of multi-level modelling [47] for transforming MyRA into a domain metamodel (RAmm in Fig. 2). Such a step is pivotal for enabling the virtual conformance checking of concrete to reference architectures.
- *Fourth step* The architect defines the concrete architecture MyA. This is done by using the ALmm created by the approach maintainer. What is more, the architect can use the graphical editor created using ALmm.
- *Fifth step* The architect using RAW defines the correspondence between the elements described in MyA and MyRA. This step is fulfilled by using weaving links since MyA and MyRA are two different models, and an additional model can be used to link them.
- *Sixth steps* Last step includes the invocation of RAV in order to check the compliance of the defined MyA with respect to RAmm obtained by MyRA. If the script does not highlight errors means that MyA is virtually conformant to MyRA. Otherwise, the approach will return a list of validation error, as well as highlighted corruption points in the visualization.

Decoupling RAmm and ALmm brings several advantages including the followings. Often, elements of a concrete architecture do not map one-to-one to elements of a RA. It means that one component in the concrete architecture can instantiate more than one component of the RA. Similarly, two components in the concrete architecture can instantiate the same component of the RA. When this happens, the decoupling mentioned above allows for establishing specific links between elements of the concrete architecture (MyA) to elements of RAs (MyRA). In the proposed approach, such links are captured by the weaving metamodel RAW. To this end, the architect uses RAW for *typing* elements of MyA to elements of MyRA. Once the architect has specified the RAW, the validation script RAV can be executed to check whether

MyA *virtually conforms* to MyRA. If MyA does not virtually conform to MyRA (red dotted line in the right part of Fig. 2), RAV informs the architect via a dedicated report containing the violated relationships. The violations are reported graphically, too. This helps the software architect get an intuitive overview of the architectural elements to be updated to achieve compliance with respect to the RA.

The interested reader can find the prototype of the implemented framework at https://github.com/gssi/RAModelingTool available for testing purposes. In the following sections, we provide a detailed description of the enabling artefacts of the proposed approach.

4.1 Generic reference architecture metamodel

GRAmm is a metamodel supporting the modelling of RAs through components and connectors [27]. In the current implementation, we refer to reference architectures that are documented in terms of a component and connector view.

Figure 3 shows the GRAmm metamodel. The starting metaclass is ReferenceArchitectureModel, which represents the whole RA and acts as a container for the RA elements and RA attributes. It contains an attribute, referenceArchitectureName, used to specify the architecture name. A ReferenceArchitectureModel is composed of one or more RAElement, each representing an element of the RA model. Such a structural containment is expressed through the raelements reference. Moreover, a ReferenceArchitectureModel is composed of zero or more RAAttribute, each characterizing an attribute related to a specified RAElement. Such a structural containment is expressed through the raattributes reference. A RAElement has two attributes name and highlighted. The former is used for specifying the name of the element, while the latter is used for representation purposes and it will be clarified later with an example. In particular, this attribute automatically highlights RA violations with respect to the defined architectural style. The reference types allows specifying zero or more roles of a component or connector that can be used within an ASV script to check an architectural style, for example publish/subscribe. RAElements can be RAComponent or RAConnector elements, where the former provides for the specification of components while the latter for connectors. RAConnector links two components through the sourceRef and targetRef associations. RAConnector has one attribute, bidirectional used to specify whether or not the communication is bidirectional. RAComponent has one attribute, mandatory, used for specifying whether a component is compulsory or not. RAComponent can contain one or more architectural elements. Such a structural containment is expressed by means of the child reference. RAElement can have zero or more RAAttribute. RAAttribute has a value type represented by the metaclass RAValueType, and the connection is expressed through the type association. A RAValueType has an attribute name to specify the name and it can be either simple, RASimpleValueType, complex RAComplexValueType or enumeration RAEnumerationValueType. A RASimpleValueType can have a text value (RATextValueType subclass), an integer value (RAIntegerValueType subclass), a real value (RARealValueType subclass), or a Boolean value (RABooleanValueType subclass). A RAComplexValueType can contain zero or more RAValueType through the structural containment child. A RAEnumerationValueType can describe a specific set (elements reference) of values (RAEnumerationItem metaclass). The value reference specifies the default value.

It is worth noting that this metamodel is kept minimal and straightforward to allow domain expert to model different domains without knowing the technical aspects of the modelled system. Moreover, we will demonstrate later that the presented metamodel is expressive enough to model different architectural domains. We have implemented GRAmm using Ecore, which is a metamodelling language part of the Eclipse Modeling Framework [48] but the presented approach can be conceptually applied to other modelling frameworks.

4.2 Architecture style validation script

RAs are composed of elements and relations among them. Typically, such elements are composed according to predefined architectural styles like, e.g. client-server, layered, publish/subscribe, pipe-and-filter, etc. [45]. In this context, ASV provides a mechanism for (i) specifying architectural style constraints and (ii) checking if a RA conforms to given architectural styles (expressed using the constraints mentioned above). ASV implements the Conf_{AS} conformance function as defined in Sect. 3. In addition, ASV can be used for specifying additional constraints (e.g. domain- and company-specific constraints) and checking the conformance of RAs to these constraints.

We have implemented ASV using the Epsilon Validation Language (EVL) [49], which is a validation language built on top of the Epsilon Object Language (EOL) [50], but in general any model validation language might be used, e.g. OCL. Listing 1 reports an excerpt of an ASV for the layered architectural style [51]. The layered style organizes architectural components with similar functionalities into layers. Each layer performs a specific role within the software system. The key principle of the layered style is isolation meaning that each layer is independent of others and has no knowledge of the inner structure of other layers. Moreover, each layer exposes interfaces for communicating with the above layers and can use interfaces provided by the bottom layers. In a nutshell, the proposed ASV checks whether MyRA has a name and if its components communicate according to the communication pattern of the layered style. It is worth noting

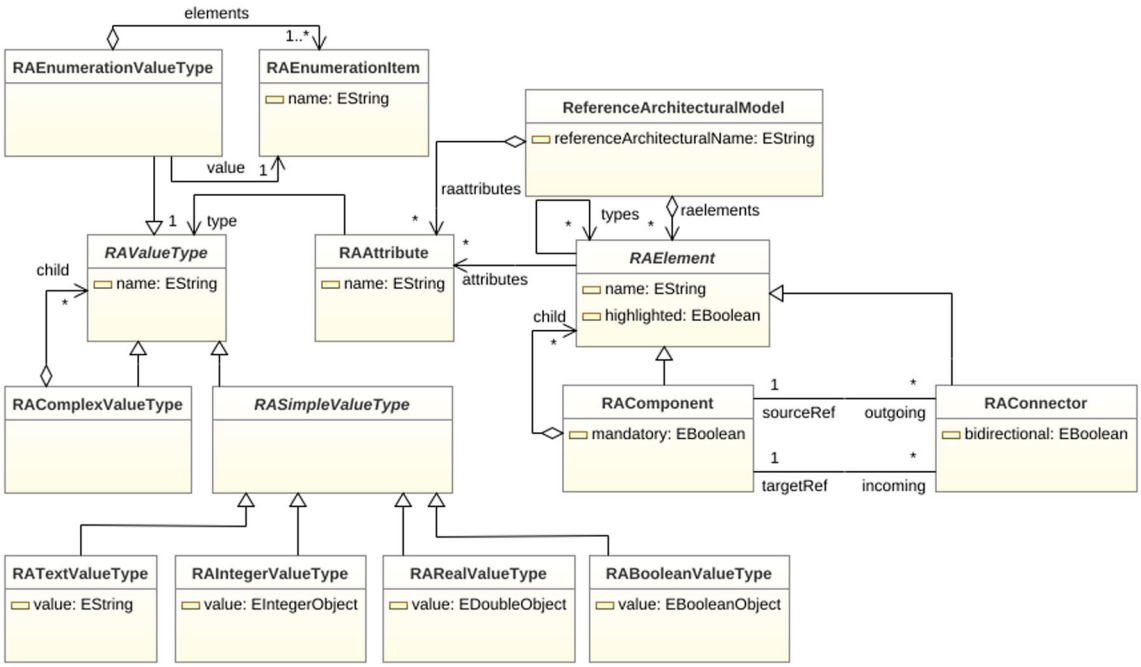


Fig. 3 Generic reference architecture metamodel (GRAmm)

that EVL also offers the functionality of a quick fix of the constraint. As can be seen in line 9 where the attribute highlighted is set to true if the constraint validation fails. We will see what this attribute does in Sect. 5, but basically, it permits to highlight in the graphical representation the problematic component in the reference architecture.

Listing 1 Excerpt of the Architectural Style Validation Script for the Layered Architectural style (ASV)

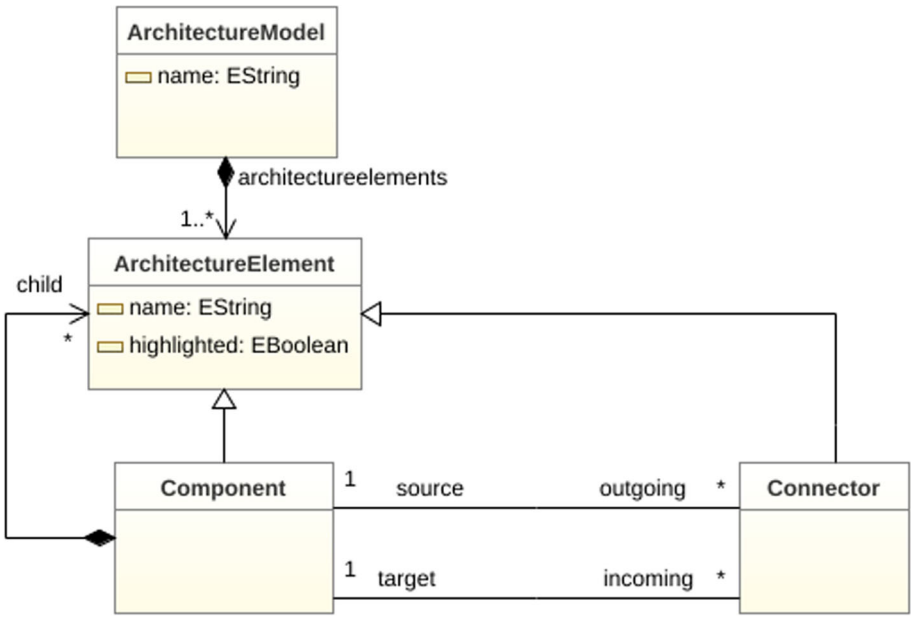
```
1   context myRA!ReferenceArchitecturalModel {
2    // Every ReferenceArchitecture must define
        a name
3     constraint HasName {
4       check : self.referenceArchitecturalName
            <> ""
5       message : "The reference architecture "
            + self + " must define a name"
6        fix {
7           title : "Fix the component"
8           do {
9               self.highlighted := true;
10          }
11      }
12    }
13  }
14   context myRa!RAComponent {
15    // No cycles are allowed
16     constraint nocycle {
17       check :  not self.allIncoming().
             includes(self)
18       message : "The component   " + self + "
            must  not be defined
            in cycles of connectors"
19    }
20  }
21  ...
```

Figure 4 shows the model concerning the publish/-subscribe architectural style [5]. It allows communication between (a) publisher(s) (Publisher component in Fig. 4) and (a) subscriber(s) (Subscriber component in Fig. 4) components asynchronously by publishing a topic through a specific protocol (e.g. MQTT). A sender, i.e. a publisher, is not aware of subscribers, and subscribers only subscribe to a specific topic. A Broker component can be used to mediate the communication among publishers and subscribers. Specifically, the broker allows subscribers to register to a specific topic and, then, when a publisher sends a message to the broker, it forwards the message to all registered subscribers related to the specific topic.

Fig. 4 Publish/Subscribe architectural style model

Listing 2 reports an excerpt of another ASV for the *publish-subscribe* architectural style. Starting from a declared component with type Broker, the script checks if the outgoing connectors point at component with type Subscriber, and, respectively, the incoming connections must come from Publisher. Moreover, by using the RAAttribute definition, it checks if two attributes are specified, i.e. the communication protocol and the topic. These attributes can also be checked at the architectural level in order to check if MyA declares the right attributes proposed in the RA, e.g. the used protocol is one specific. If this validation rule fails, the component will be highlighted to be identified in the visual representation. This example can be easily enriched with additional checks, but we kept it simple for the sake of understandability.

Fig. 5 The AL metamodel



Listing 2 Excerpt of the Architectural Style Validation Script for Publish/Subscribe Architectural style (ASV)

```
1   context mymodel!RAComponent {
2       constraint pubsub {
3           guard:  self.types.name.includes("
                Broker")
4           check :
5           self.outgoing.targetRef.types.
                flatten().forAll(t|t.name="
                Subscriber")
                and
6           self.incoming.sourceRef.types.
                flatten().forAll(t|t.name="
                Publisher")
                and
7           self.attributes.name.includesAll(
                Sequence{"communicationprotocol"
                ,"topic"})
8           message : "The component  "+self.
                name+" is a broker
                and must have outgoing
                connectors to Subscribers
                and incoming from Publisher"
9           fix {
10           title : "Fix the components"
11           do {
12              self.highlighted := true;
13          }
14      }
15   }
16  }
```

4.3 Model promotion transformation

T uses the principle of *promotion* [46] in the field of multi-level modelling [47] for transforming a model of a RA (MyRA in Fig. 2) into a domain metamodel (RAmm in Fig. 2). This is a crucial step for enabling the virtual conformance check.

We rely on the notion of compliance [52] between MyA and MyRA, which we implemented as virtual conformance. The conformance relationship in MDE defines a typing relationship between objects and their types. Conceptually, we use reference architectures as templates for concrete architectures, and for this reason, it seems natural to promote objects into instantiable meta-classes. In fact, MyA is defined in terms of an architectural language, and in addition to that, we should also check its compliance concerning MyRA. From a technological point of view, promoting a MyRA model to a metamodel, and in our implementation to an ecore [48] model, enables other possible applications to make the approach more usable. For instance, an editor for designing architectures with respect to the architectural language could be provided. This editor may be based on Sirius [53] or Eugenia [54] and may allow checking the conformance of the defined architecture both with the architectural language and the reference architecture. Concrete syntax definitions work with domain models, and for this reason, we need to have a metamodel frozen with static types to enable conformance and virtual conformance checks of the defined architectural model.

Listing 3 reports an excerpt of the T transformation.

Listing 3 Excerpt of the Model Promotion Transformation (T)

```
1  ...
2  rule RefArchModel
3  transform s : ra!ReferenceArchitecturalModel
4    to p : ecore!EPackage, c:ecore!EClass {
5    p.name = s.referenceArchitecturalName;
6    p.eClassifiers::=s.raelements.select(c|c.
         isTypeOf(RAComponent));
7    c.name = s.referenceArchitecturalName;
8    for (comp in s.raelements.select(c|c.
         isTypeOf(RAComponent) and c.mandatory)
         ) {
9      var  ref: ecore!EReference=new ecore!
           EReference();
10     ref.name=comp.name.toLowerCase()+'s';
11     ref.eType::=comp;
12     ref.lowerBound=1;
13     c.ereferences.add(ref);
14   }
15   p.eClassifiers.add(c);
16   ...
17 }
18 rule RAComponent
19 transform s: ra!RAComponent
20 to c: ecore!EClass {
21   c.name=s.name;
22   c.eStructuralFeatures::=s.outgoing;
23 }
24 rule Connector
25 transform s: ra!RAConnector
26 to r: ecore!EReference {
27   r.name=s.name;
28   r.eType::=s.target;
29 }
30 ...
```

We have developed T using the Epsilon Transformation Language (ETL), which is a hybrid, rule-based model-to-model transformation language built on top of EOL [55]. The main rules composing T are as follows. Starting from a model of a RA (ra variable), the transformation creates a metamodel containing a starting package with the name of the RA (line 4). For each RAComponent in the model of the RA, T creates a new instance of EClass. If a RAComponent is marked as mandatory, then T translates it as a compulsory instance for the metamodel (lines 8–14). This will force the modeller to add (at least) one element corresponding to that RAComponent. For each RAConnector in the model of the RA, T creates a new reference accordingly. For simplicity, we left out of the discussion the attributes, but they are defined in the same way. It is worth noting that this transformation is executed in a stage in which MyRA is well-defined and also passed a validation check. The architect should not come back to the reference architecture definition as long as the metamodel is obtained.

4.4 AL metamodel

ALmm is a metamodel for the specification of concrete architectures as a crucial step towards the virtual conformance check of concrete to reference architectures. Within the proposed approach, the architect has several possibilities for modelling concrete architectures. In case of ALs are already used, or the architect is familiar with existing ALs (*e.g.* EAST-ADL [56] or RCM [57] in the automotive domain), the concrete architecture can be modelled using any of these ALs. A technical bridge can be easily implemented by specifying model transformations, where the source metamodel is the existing AL and the target is ALmm. This task requires establishing the existing mappings between the ALs, and define transformation rules for the translation. Additionally, if no ALs are used, the architect can define her own AL through a metamodel or alternatively can use the provided ALmm, which we have defined, bearing in mind simplicity and reusability across different domains.

Figure 5 shows the main concepts of the ALmm metamodel. The starting metaclass is ArchitectureModel, which act as a container for the whole concrete architecture. It contains an attribute, name, used to specify the architecture name. ArchitectureModel is composed of one or more ArchitectureElements, which represent elements of the architecture. Such containment is expressed employing the architectureelements reference. An ArchitectureElement can be either a Connector or Component and also in this case can be highlighted in the graphical representation. The latter provides for the specification of components while the former for connectors. A connector links two components through the source and target associations. Also, in this case, we have implemented ALmm using Ecore.

The architectural models specified instantiating this metamodel can be automatically rendered in a dedicated view, graphically showing the declared components and connectors. These views are automatically generated through code generation (expressed with EGL [58] template) and visualized through Picto [59], an Epsilon plugin allowing synchronized views of the models.

4.5 Expressing the relation between MyA and MyRA

RAW is a metamodel for typing elements of concrete architectures into elements of RAs. Such typing, which might be seen as a special instance of so-called model weaving [60], is a way to establish fine-grained correspondences between elements from different models so as to use these correspondences for validation purposes. In our case, such correspondences are called relations and type elements of MyA to elements of RAmm.

Figure 6 provides a graphical representation of RAW. The main metaclass is ImplementationModel and acts as a container for the relationships. To this end, it can contain one or more RAALRelation elements where each of this element links one or more instances of an EClass, generated from RAmm, to one or more instances of Component in MyA. We have implemented RAW using Ecore.

This weaving metamodel allows linking model elements of different models. As can be seen from the metamodel in Fig. 6, the two references relate to different metamodels.

Fig. 6 RA tagging weaving metamodel (RAW)

This task of linking elements in models conform to different metamodels would be problematic and not very usable. For this reason, in our framework, we use a customization of the Modelink [61] plugin, which is an Epsilon module allowing us to have a split-view of the models, and using drag and drop, the user can actually link models through the specified weaving metamodel.

4.6 RA validation script

RAV allows for checking the virtual conformance of concrete to reference architectures, as defined in the $\text{Conf}_{\text{MyRA}}$ function (see Sect. 3). Listing 4 describes an excerpt of RAV.

Listing 4 Excerpt of the Generated RA Validation Script (RAV)

```
1   import "validation-scripts.eol";
2   context weaving!RAALRelation {
3     constraint relationType {
4       check :  self.checkRelation()
5       message: "component: " + self.component
            .name + " does
            not respect the RA rules"
6       fix {
7       title : "Fix "+self.component.name
8           do {
9               for (c  in self.component) {
10                  c.highlighted:=true;
11              }
12          }
13      }
14    }
15  }
16  context weaving!ImplementationModel {
17    constraint MandatoryComponents {
18      check :  self.checkMandatoryComponents
            ()
19      message: "Not all the mandatory
            components  in the RA are used
            in the architecture"
20    }
21  }
22  ...
```

The inputs of RAV are RAW, RAmm and MyA while the main checks of RAV are as follows. RAV checks that all the mandatory RA components are instantiated from components of concrete architectures (line 10). RAV checks that all the relationships among components of concrete architectures follow the patterns described in the related RA (line 4). Moreover, this constraint check also highlights in the declared architectural model if a validation error is present. This script acts on the highlighted attribute and allows the generated views in Picto to immediately make evident the violations of the architectural model with respect to the RA. This is supported by the fix construct declared at lines 6–13. We remind here that this script is generic and it is RA-independent and can be used to validate all the RAs, provided that RAW is correctly composed. We have implemented RAV using EVL and declared the checks mentioned above in helper libraries called validation-scripts.eol. These script are organized in multiple EVL files that the user can import when selecting the needed architectural style.

5 Evaluation

The main goal of this evaluation is to assess the proposed approach in terms of its (i) *applicability*, (ii) *effectiveness*, and (iii) *generalizability*. Each illustrative example covers one of the three characteristics mentioned above. We have conducted these experiments following a rigorous research method and validated them using a mix of observational and experimental methods comprising three different use cases from two different application domains, i.e. web browsers and automotive. We outline that in order to select the candidate case studies, we have conducted a literature search to find papers including examples clearly defining an architectural model and the corresponding reference architecture. We have limited the search to examples in which the compliance of the two architectures is thoroughly described. In this task, we have excluded examples in which one of the architectures is not well defined, or the compliance cannot be derived explicitly.

In the first use case, we leverage the work by Grosskurth and Godfrey [24] presenting a reference architecture for web browsers and one concrete architecture. This example demonstrates that the proposed approach is *applicable* on real software products. In the second illustrative example, we demonstrate that the proposed approach is *effective* in detecting different types of conformance issues between concrete and reference architectures. To this end, we start from the previous example on web browsers and, using a model mutator [25]; we produce several faulty mutated concrete architectures. Then, we check that our approach is able to detect all the conformance issues introduced by the mutations. In the third illustrative example, we build on the work by Maple et al. [26] on a cloud RA for autonomous vehicles. Specifically, we apply our approach to model (i) the cloud RA for autonomous vehicles and (ii) a concrete architecture for a driver-less valet parking vehicle. The last illustrative example is another cloud solution from the automotive domain and comes from a reference architecture and concrete solution from AWS Architecture Center. In particular, we used this case study because it leverages the publish/subscribe architectural style. These examples demonstrate that the proposed approach is applicable in different application domains, thus indicating its *generalizability*.

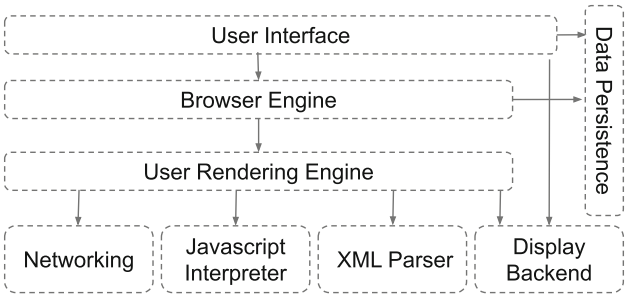


Fig. 7 Reference architecture for web browsers [24]

5.1 Architecting web browsers

The work by Grosskurth and Godfrey [24] describes a RA for web browsers and analyses a set of concrete architectures of commercial web browsers like Mozilla Firefox, Apple Safari, etc. We use our approach for (i) representing the RA of web browser and (ii) modelling and checking the virtual conformance of the Mozilla Firefox architecture to the RA for web browsers.

Figure 7 shows the RA for web browsers. It is composed of eight components and nine connectors. The User Interface component is the layer between the end-user and the Browser Engine and provides features such as toolbars, visual page-load progress, etc. It may be integrated with desktop environments for providing browser sessions management or communication with other desktop applications. The Browser Engine is responsible for loading a given Uniform Resource Locator (URL). It supports primitive browsing actions, provides features for viewing various aspects of the browsing session, such as current page load progress, and allows the querying and manipulation of settings of User Rendering Engine. The Browser Engine component provides a high-level interface to the User Rendering Engine. The User Rendering Engine component provides a visual representation of a given URL. To this end, it displays HyperText Markup Language (HTML), Extensible Markup Language (XML), Cascading Style Sheets (CSS), and embedded content (e.g. images). The Networking component implements file transfer protocols, translates among different character sets, resolves Multipurpose Internet Mail Extensions (MIME) media and implements a cache of recently retrieved resources. The Javascript Interpreter evaluates JavaScript code. The XML Parser component is responsible for parsing XML documents into a Document Object Model (DOM). The Display Backend component provides drawing and windowing primitives for a set of user interface widgets and fonts. Finally, the Data Persistence component stores on disks data associated with the browsing sessions.

According to our proposed approach, the first step is to represent the above RA as a model conforming to the GRAmm metamodel. Figure 8 shows the automatically generated graphical representation of the modelled RA ①, while ② shows its serialization. Using GRAmm we were able to represent the whole original RA composing of eight components and nine connectors in negligible time.

The second step of the approach is to check if the modelled RA follows the architectural style guidelines. In this example, we applied the guidelines related to the layered architectural style described in Sect. 4.2. As shown in ③, the execution of the ASV script produces four errors, so the modelled RA does not respect the layered architectural style. In particular, the first three errors indicate that the modelled architecture has wrong connections (red arrows in ①), while the fourth error indicates that the modelled architecture is unnamed. ④ also shows an additional view (provided by the generated graphical representation of the RA) where the user can explore singularly the components of the RA and better spot the errors, e.g. the highlighted connector between Browser Engine and User Interface. Moreover, the general view in ① demonstrates how the errors in the RA can be easily spotted in its graphical representation, where all the elements involved in the validation errors are highlighted. In the second round of modelling, we fixed all validation errors, leading to a final RA supporting all the constraints of the layered architectural style.

Once all the validation errors are fixed, we run the model transformation T to *promote* the modelled RA into a metamodel. The output produced by this transformation is shown in Fig. 9. For instance, the components UserInterface and DataPersistence together with the connector DataPersistence have been promoted to metaclasses and relation. For the sake of readability, we hide the metaclass RAWebBrowserModel as it has composition relations towards all the remaining metaclasses. However, it is important to note that these relations are crucial as they ensure the mandatory presence of components. At this point in the approach, we have a RA model for web browser, which has been promoted to a metamodel. Hence, we have a metamodel of a RA for web browsers.

The next step of the approach is creating a model representing a concrete architecture of a web browser. We decided to focus our attention on the Mozilla Firefox browser since it was already modelled in details in [24], thus allowing us to double-check the correctness of our model. Figure 10 shows the architecture of Mozilla Firefox (black solid lines and arrows) according to [24].

As discussed in Sect. 3, concrete architecture can be formalized through ALmm or any other ADL. In this example, we use ALmm for modelling the Firefox architecture. Using ALmm we were able to model the original Firefox architecture in [24].

The next step of the approach is to use RAW to tag elements of the concrete Mozilla Firefox architecture to ele-

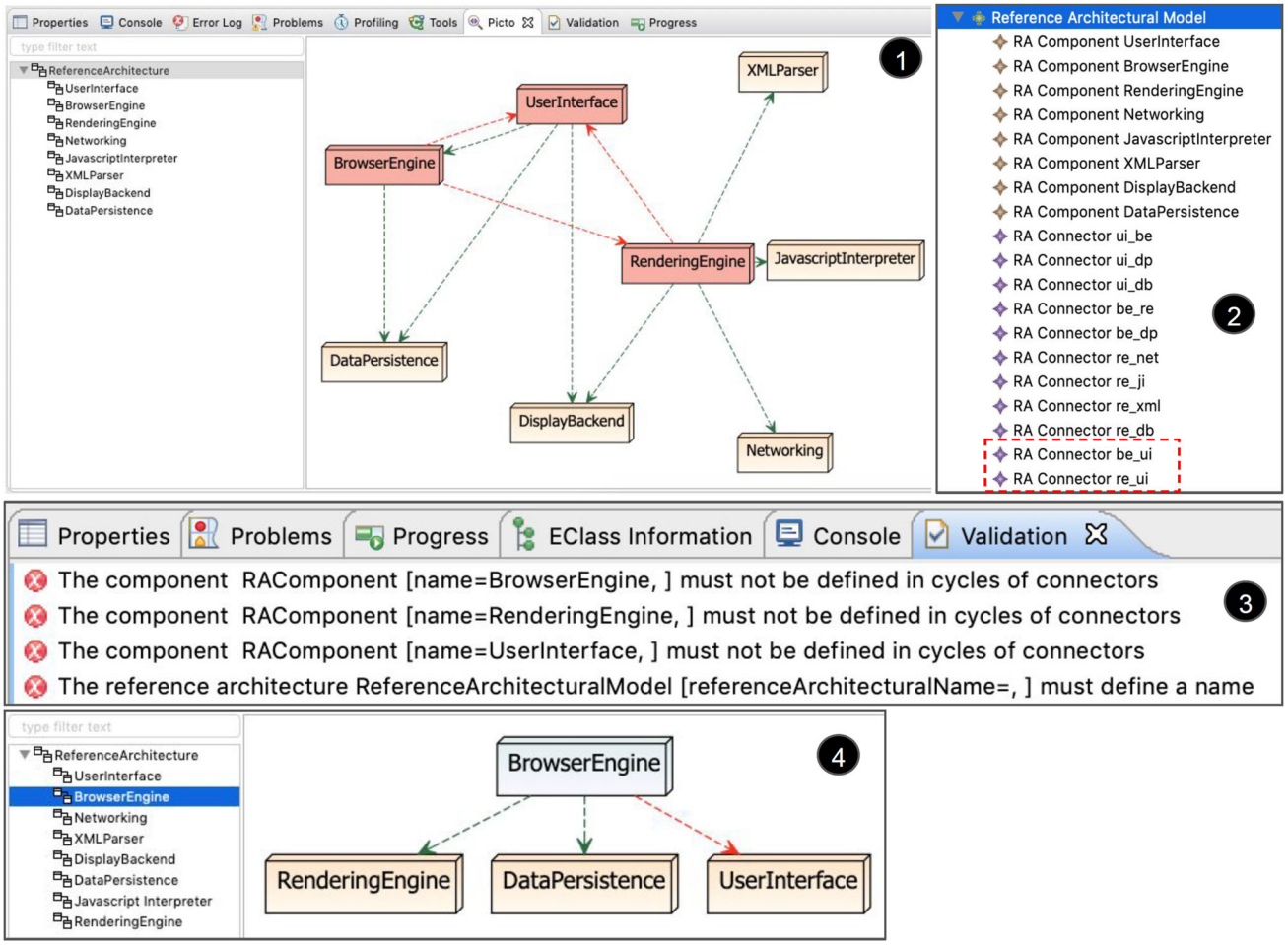


Fig. 8 Modelled reference architecture for web browsers and validation results

ments of the generated RA for web browsers. In our case, we have followed the relations specified in [24] and graphically represented in Fig. 10. In Fig. 10 the elements of the Mozilla Firefox architecture are represented with solid black lines, whereas the elements of the RA for web browsers are described as grey dashed lines. These relations mean that, e.g. the concrete Mozilla Firefox component Gecko is tagged with the RA Browser Engine User Rendering Engine components. In the proposed approach, such relations can be formally captured using RAW and will be the main artefact for validating the Mozilla Firefox architecture against its RA.

Figure 11 shows how these relations are captured in the tool implementing our approach. In particular, ① contains a tree-based representation of the Mozilla Firefox architecture, which conforms to the ALmm metamodel, and ⑤ graphically renders it. ③ depicts a three-based representation of the RA for web browsers, and ② represents the RAW model, hence the model containing the type-of relationships between the Mozilla Firefox architecture and the RA for web browsers. For example, the first element of the RAW model (see the selected link in ②) outlines the connection between the UserInterface and UI Toolkit components in the Mozilla Firefox architecture and the UserInterface component in the RA for web browsers. When all the relations are specified, the next step is the execution of the RAV. To show how such a script works, we have purposely specified a wrong connector in Fig. 11. Such a connector (named wrong and highlighted with a dotted red rectangle in Fig. 11 links the component Gecko to the component UI Toolkit (XPFE) (see ⑤). However, a connection is not allowed in the RA since no connectors are defined between those two components. The RAV raises the error by highlighting in red the problematic elements in the architecture, i.e. UIToolkit and Gecko in ④. Fixing the error highlighted in Fig. 11 requires the removal of the connector from Gecko to the UI Toolkit. Only when that connector is removed, the model representing the Mozilla Firefox architecture passes the evaluation and hence it *virtually* conforms to the RA for web browsers. It is important to note that, within the proposed approach, fixing the mistake mentioned above has required negligible time. However, without the support of the proposed approach, detecting and

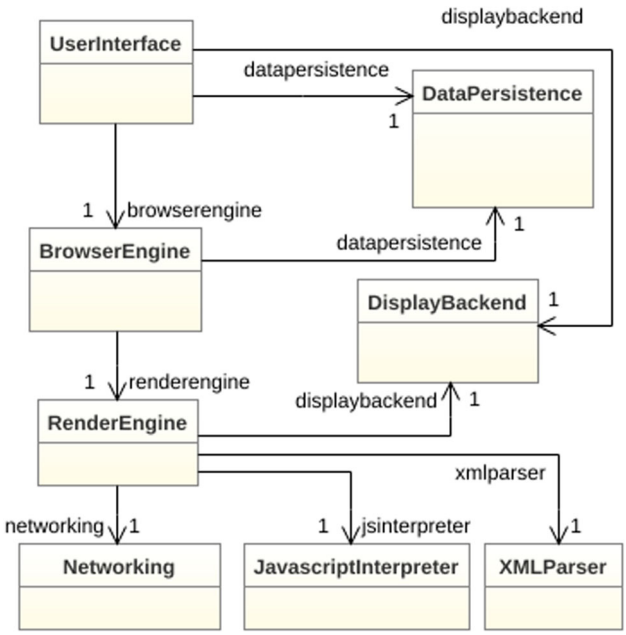


Fig. 9 Generated metamodel for the reference architecture (RAmm) in Fig. 7

correcting the issue would have required substantial effort as the architect was required to manually compare the concrete and reference architectures, their components and connectors.

5.2 Error injection to test the effectiveness of the approach

In this section, we evaluate the *effectiveness* of the proposed approach in identifying conformance issues between reference and concrete architectures. To this end, we build on the web browser example presented in Sect. 5.1.

In order to cover as many potential issues as possible, we exploit the model mutation technique [62]. Such a technique allows for generating several mutated models with low effort and increased control over their characteristics in terms of structural constraints. In our case, mutations are automatically generated using the model mutator by Gómez-Abajo et al. [63] and represent mutated concrete architectures for which we check their virtual conformance. The mutation process is applied to both the web browsers RA in Fig. 8 and the concrete architecture of the Mozilla Firefox system in Fig. 11.

The mutations on the web browsers RA are as follows:

RA1: Mark UserInterface as mandatory;
RA2: Remove the connector RenderingEngine → Networking and added connector Networking → JavascriptInterpreter;
RA3: Remove the connector RenderingEngine → DisplayBackend;
RA4: Remove the name from the RA and added connector Networking → UserInterface.

Similarly, the mutations on the Mozilla Firefox architecture are as follows:

A1: Remove component Necko, changed connector from Gecko → Necko to Gecko → Security, removed connector Necko → Security;
A2: Remove connector Toolkit → Gecko, removed connector Gecko → Necko, removed connector Gecko → SpiderMonkey;
A3: Add anewComponent, removed component UserInterface, changed connector UserInterface → ToolKit to anewComponent → Toolkit;
A4: Change connector Necko → Security to Necko → SpiderMonkey;
A5: Remove component UserInterface, removed connector Gecko → GTKAdapter, removed connector UserInterface → Toolkit.

The chosen mutations have been selected to cover as many corner cases as possible and apply random operations on components and connectors to test the approach from different angles. Combining the above mutations, we obtain 20 different scenarios (see Table 1). The first column shows the combinations between the applied mutations and acts as a unique identifier for each combination. The second column shows whether the ASV validation of the RA succeeds (✓) or fails (✗). The third column shows whether the RAV validation between the concrete and reference architecture succeeds (✓) or fails (✗). In the fourth column, we report the error message produced by our tool (if any), and the last column contains the validation results in terms of *accuracy* (true(T)/false(F) positives(P)/negatives(N)).

For each of these scenarios, we update the RAW model by simply fixing the dangling elements. This operation is needed since the approach works with RAW, and applying mutations on the existing MyA and MyRA may invalidate the weaving model, resulting in dangling connectors or missing data types, for instance, in case of the removal of a component. Finally, we run our RAV validation script and ask an expert to assess the results. The expert involved in this experiment has been selected among our network of collaborators. The expert has been involved in both research and development activities for 11 years, and their technical background involves modelling tools and software architecture.

As shown in the last rows of Table 1, the model mutator generated violations in the RA in 5 mutations, violations in the concrete architecture in 8 mutations, while the remaining 7 mutations did not contain any violation, both in the RA and

in the concrete architecture. In this evaluation, we consider the assessment provided by the expert as the ground truth. So, based on the evaluation provided by the expert, the result of a single combination of mutations can be one of the following:

- TP (True Positive): both the approach and the expert detect a violation in either the RA or the concrete architecture.
- TN (True Negative): both the approach and the expert confirm the validity of both the RA or the concrete architecture.
- FP (False Positive): the approach detects a violation in either the RA or the concrete architecture, but the expert does not.
- FN (False Negative): the approach confirms the validity of both the RA and the concrete architecture, but the expert detects a violation.

It is important to note that we marked the results of our validation as a True Positive (TP) only when the expert detected *exactly* the same violation as the approach.

From this experiment, we have concluded that the approach is able to detect the violations expressed at ASV-level or RAV-level in all the generated mutations (13 TPs). If the mutations did not induce a violation in the reference architecture and/or in the architectural model, the approach could confirm the virtual conformance relationship (8 TNs). Finally, the validation revealed that the approach is aligned with the indication of the expert both when the expert detected a violation (i.e. no false negatives) and when the expert did not (i.e. no false positives).

5.3 Architecting vehicle cloud systems

The work by Maple et al. [26] describes a cloud RA for connected and autonomous vehicles for attack surface analysis. Besides, the work describes the concrete architecture of a driver-less valet parking system. To demonstrate that our approach can be applied to different domains, we model the cloud RA and concrete model presented by Maple et al. and check that the concrete architecture for driver-less valet parking conforms to the cloud RA. Figure 12 shows the cloud RA for connected and autonomous vehicles for attack surface analysis (grey dotted lines) and the concrete architecture of a driver-less valet parking system (solid black lines).

The RA is composed of ten components and twelve connectors. Vehicles components represent autonomous vehicles and are connected to the Edge component. The Edge component allows interaction with the vehicle's devices and peripherals (Device and Peripherals component), third-party clouds (3rd Party Clouds component), and with the Communication component. The Communication component is

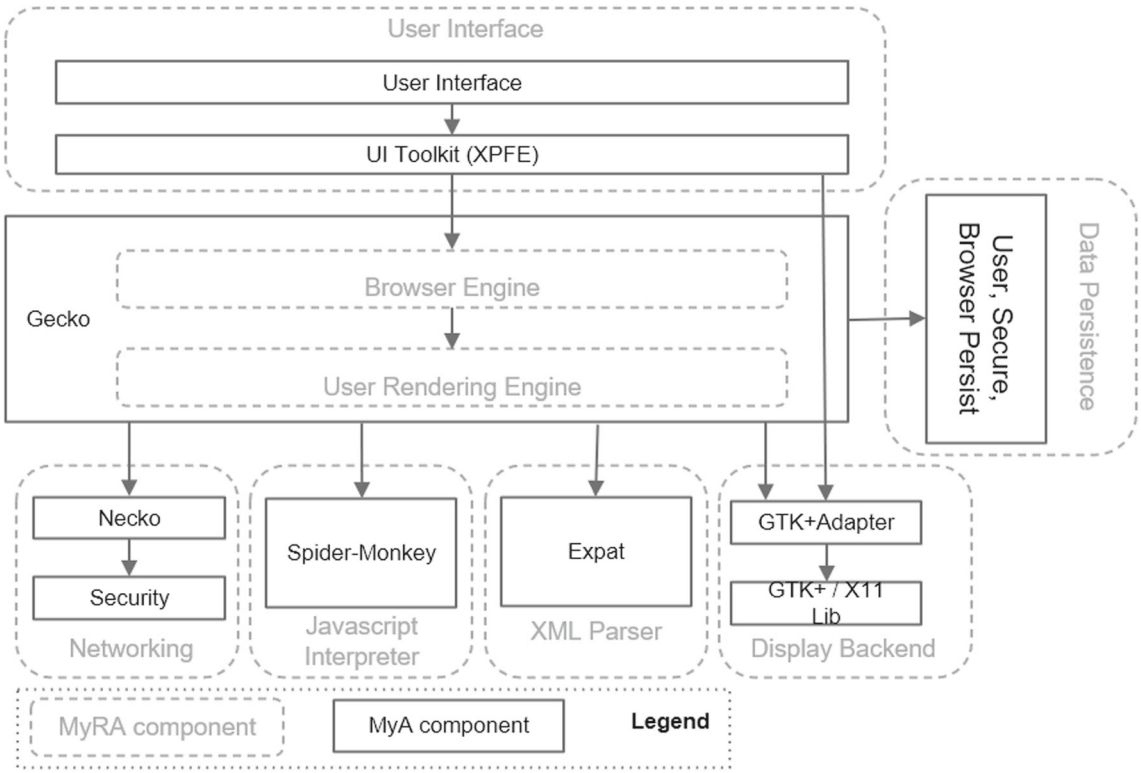


Fig. 10 Mozilla Firefox architecture (A) [24]

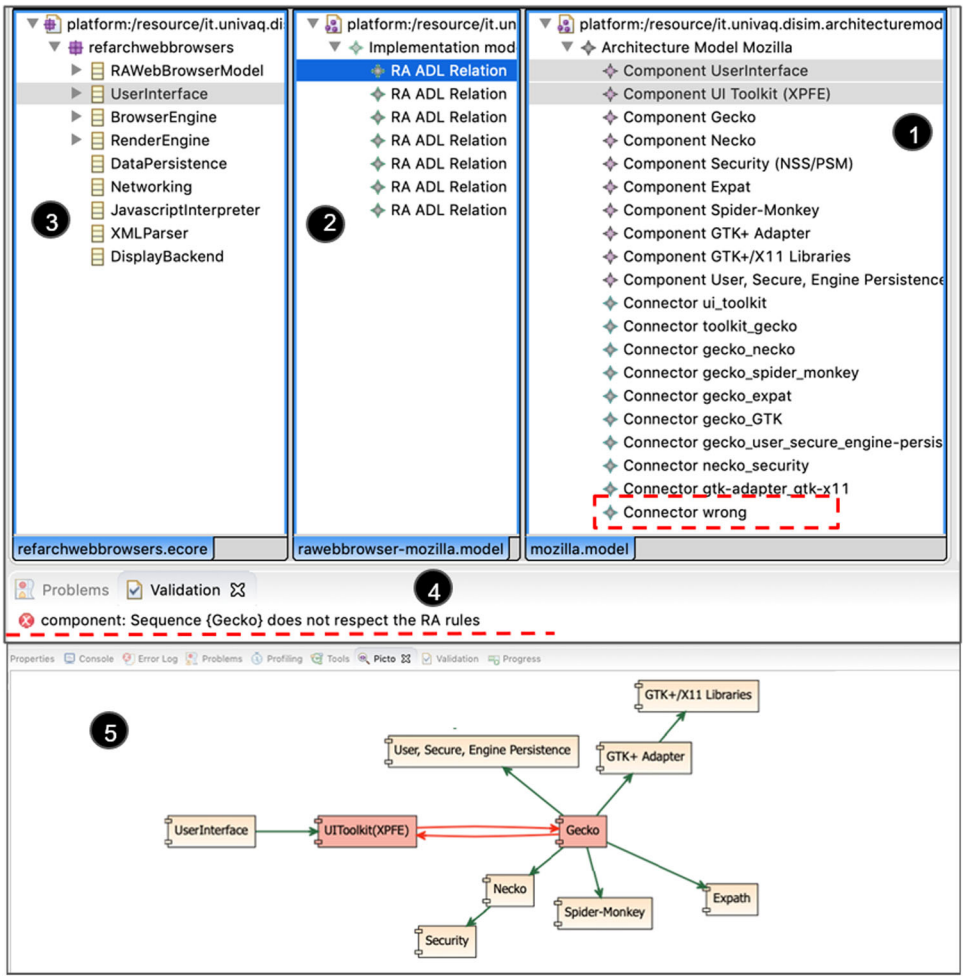


Fig. 11 Weaving editor for the RA Modelling of Mozilla Firefox example

responsible for managing vehicle communication with the cloud. It is envisioned to leverage multiple gateways and provide protection against external attacks, too. The Data Storage component is in charge of storing the vehicle data, including the firmware and software used to run the car, maps and navigation information, files for the infotainment system, etc. This data will be physically distributed across many different data centres and replicated to ensure integrity and availability under hardware failures. The Data Analysis component is responsible for analysing either the data obtained from external sources or the data stored in the vehicle for different purposes. An example of such analyses is analysing historical data coming from all cars to predict traffic patterns for balancing a load of road networks when a vehicle requests a route. The Microservices provides vehicle functionalities via a collection of loosely coupled services, while the Application Programming Interfaces (APIs) component is a collection of interfaces of services exposed by the vehicle. Eventually, the Monitoring component verifies that the vehicle is functioning correctly.

Figure 13 ① shows a screenshot of the tool implementing our approach where we modelled the cloud RA for autonomous vehicles using GRAmm. As for the RA for web browsers, we were able to model the whole original RA for connected and autonomous vehicles in negligible time.

As Maple et al. defined the cloud RA for autonomous vehicles without referring to any specific architectural style [26], we do not use ASV for checking the conformance of the modelled RA to a particular architectural style. At this point, we call T on the modelled cloud RA. We obtain the metamodel representing the concept of the RA at the metamodelling level (see ②). ④ shows a screenshot of our tool containing the model of the concrete architecture of the driver-less valet parking system developed with our approach (represented with the proposed AL). All the elements depicted in Fig. 12, concerning the architecture and reference architecture proposed, have been reported using our approach. ③ shows the RAW model defining the relationships [26] between the concrete architecture of the valet parking and the cloud RA for autonomous vehicles, following what has been described in [26]. With the presented approach, we also defined ⑤ the architecture defined in Fig. 12 with the modelling tool. Eventually, we have run the RAV validation script, which did not identify any conformance error. Hence, the driver-less valet

Table 1 Validation results with mutated models

Combinations	ASV	RAV	Errors	Result
RA0 A0 (seed)	✓	✓	–	TN
RA1 A1	✓	✓	–	TN
RA2 A1	✓	✗	Component Gecko does not respect the RA rules	TP
RA3 A1	✓	✓	–	TN
RA4 A1	✗	–	The reference architecture must define a name, the component(s): Networking, User Interface must not be defined in cycles of connectors	TP
RA1 A2	✓	✗	Component(s): UI Toolkit (XPFE), Gecko does(do) not respect the RA rules	TP
RA2 A2	✓	✗	Component(s): UI Toolkit (XPFE), Gecko does(do) not respect the RA rules	TP
RA3 A2	✓	✗	Component(s): UI Toolkit (XPFE), Gecko does(do) not respect the RA rules	TP
RA4 A2	✗	–	The reference architecture must define a name, the component(s): Networking, User Interface must not be defined in cycles of connectors	TP
RA1 A3	✓	✓	–	TN
RA2 A3	✓	✗	Component(s): SpiderMonkey does(do) not respect the RA rules	TP
RA3 A3	✓	✓	–	TN
RA4 A3	✗	–	The reference architecture must define a name, the component(s): Networking, User Interface must not be defined in cycles of connectors	TP
RA1 A4	✓	✓	–	TN
RA2 A4	✓	✓	–	TN
RA3 A4	✓	✗	Component(s): Necko, GTK+ Adapter does(do) not respect the RA rules	TP
RA4 A4	✗	–	The reference architecture must define a name, the component(s): Networking, User Interface must not be defined in cycles of connectors	TP
RA1 A5	✓	✗	Component(s): UserInterface, GTK-Adapter does(do) not respect the RA rules	TP
RA2 A5	✓	✗	Component(s): GTK-Adapter, Necko does(do) not respect the RA rules	TP
RA3 A5	✓	✓	-	TN
RA4 A5	✗	–	The reference architecture must define a name, the component(s): Networking, User Interface must not be defined in cycles of connectors	TP
	–	–	**TP**	13
#violations	5	–	**TN**	8
#violations	–	8	**FP**	0
	–	–	**FN**	0

parking system architecture *virtually* is well-formed with respect to the cloud reference architecture for connected and autonomous vehicles for attack surface analysis. This makes us reasonably confident about the correctness of the structure of the modelled architecture concerning the identified constraints of the domain of the autonomous vehicle.

5.4 AWS connected mobility architecture

We applied our approach to a reference architecture and a concrete architecture solution borrowed from the AWS Architecture Center. It contains more than 270 reference architectures, best practices, and AWS cloud reference solutions for different domains, such as automotive, healthcare, and manufacturing. In particular, Fig. 14 depicts the AWS Connected Mobility reference architecture (grey dot-

ted lines)[9] and the architecture solution (represented with AWS Architecture Icons[10]) that allows collecting, processing, analysing, and operating on connected vehicle data.

The RA is composed of thirteen components and eighteen connectors. The RA uses the Publish/Subscribe architectural style (see Fig. 4). In particular, the Connected Vehicle component acts as a Publisher, and it represents a connected vehicle that sends information about its mobility. The IoT Core component identifies the gateway from the connected vehicles to the others components and operates as a Broker. Using some business rules, it dispatches the incoming messages to the related components, i.e. Subscribers. In particular, the Raw Telemetry component stores all the incoming messages. The Anomaly Detection, Driver Score, Diagnostic Codes, and Drivetrain Telemetry and GPS components analyse and process messages to detect anomalies, calculate the driver's score, identify diagnostic codes, and extract drivetrain and GPS data, respectively. The Vehicle Simulators component simulates trip routes and telemetry data. The AWS Connected Device Framework component includes several modules to manage the connected vehicles in their different lifecycle phases. The IoT Device Management component allows connecting the vehicles with the AWS Connected Device Framework. The Fleet Management UI, Fleet Manager Data Storage and Access, and Automotive Faces components are used by the Fleet Manager user to configure and maintain the fleet.

Figure 15 reports the tool in action on the subject architecture and RA in Fig. 14. As can be seen from RAW, the elements of the architecture have been tagged with the relative component types of the RA. The graphical representation generated by the tool highlights that the AWS IoT SDK presents some problems. Indeed, a missing connector between a component that acts as a Publisher and the one acting as a Broker is missing, i.e. between the AWS IoT SDK and the AWS IoT Core. If we restore the missing connector, the virtual conformance is reached, and MyA will be compliant with MyRA in this example.

5.5 Threats to validity

In the following we discuss the threats to validity of the three previously discussed illustrative examples according to the Campbell and Stanley categorization [64].

5.5.1 External validity

One possible threat to external validity is the representativity of the proposed metamodels and the approach's applicability. To mitigate such an issue, we have applied the proposed approach in two different domains, i.e. web browsers and automotive systems. It should be noted that the two chosen application domains are very different in terms of their implied technological stacks and the technical background of their involved stakeholders (e.g. automotive engineers vs web developers). We outline that one of our case studies has been borrowed from the Amazon AWS RAs repository and that the example we have tested in our approach reflects a real case study reported on their website. This example has been selected for multiple reasons: first, it represents a medium-sized RA with respect to the existing examples that can be clearly identified online; second, it demonstrates that our approach can be applied to real-world examples. Indeed, the compliance check and the concrete architecture we represented confirm that the reference implementation on the AWS website is effectively compliant with the RA. This verification has been possible since we faithfully represented what is hosted on the AWS website in our approach. Nevertheless, we are aware that further applications in additional application domains are needed to provide objective evidence about the applicability of the approach in the context of any application domain. In order to mitigate this potential limitation, we are planning a series of industrial case studies across several application domains where the authors are already active and have several industrial collaborations, such as robotics, mobile apps development, microservice-based systems, etc.

Moreover, having a fixed set of mutated models may not be sufficient to provide objective evidence about the effectiveness of the approach. We mitigated this potential threat to validity by having a relatively high number of combinations of mutated models (i.e. 20 different scenarios) and having a domain expert assess whether those scenarios result in a violation or full conformance of the concrete architectures against them their corresponding reference architecture. We did this procedure in the context of the web browser domain only due to limitations in available resources. For future work, we plan to expand this part of the evaluation of the proposed approach by considering multiple domains and external domain experts with varying degrees of experience in the field.

Finally, due to the low number of involved concrete architectures, we did not perform any statistical analysis of the obtained results, and we just reported the number of true/-false positives/negatives. This limits the statistical power of the performed evaluation, in turn limiting the generalizability of the obtained results when considering the effectiveness of the proposed approach.

[9] https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/aws-connected-mobility-ra.pdf?did=wp_card&trk=wp_card.

[10] https://aws.amazon.com/architecture/icons/?nc1=h_ls.

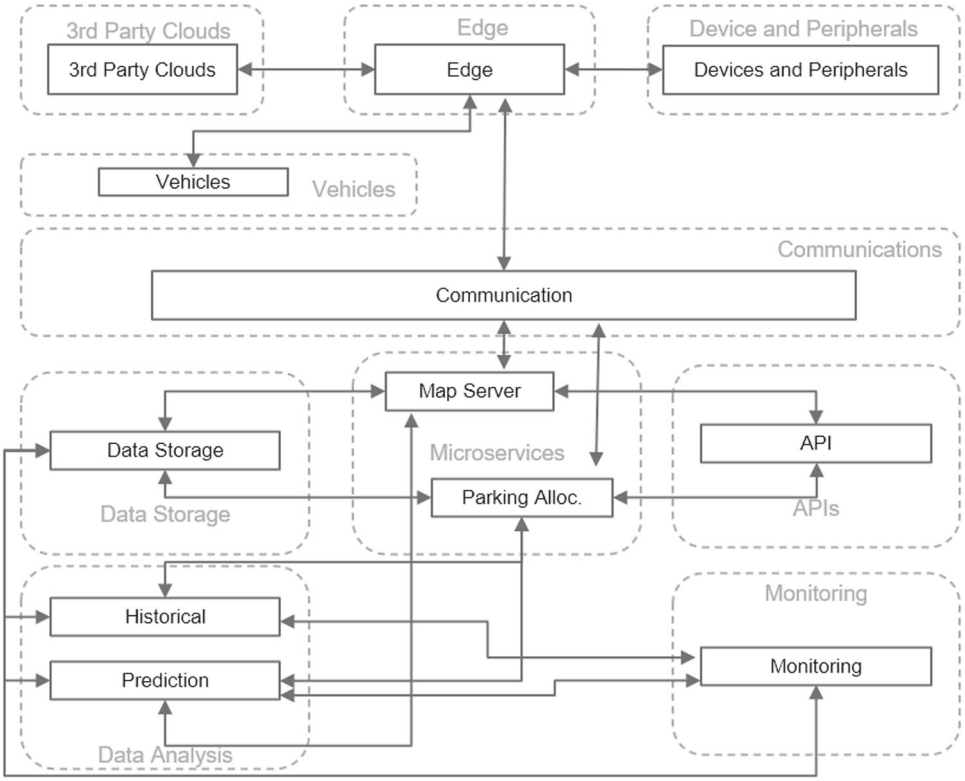


Fig. 12 Cloud reference architecture for connected and autonomous vehicles for attack surface analysis (grey dotted lines) and concrete architecture of a driver-less valet parking system (black solid lines) [26]

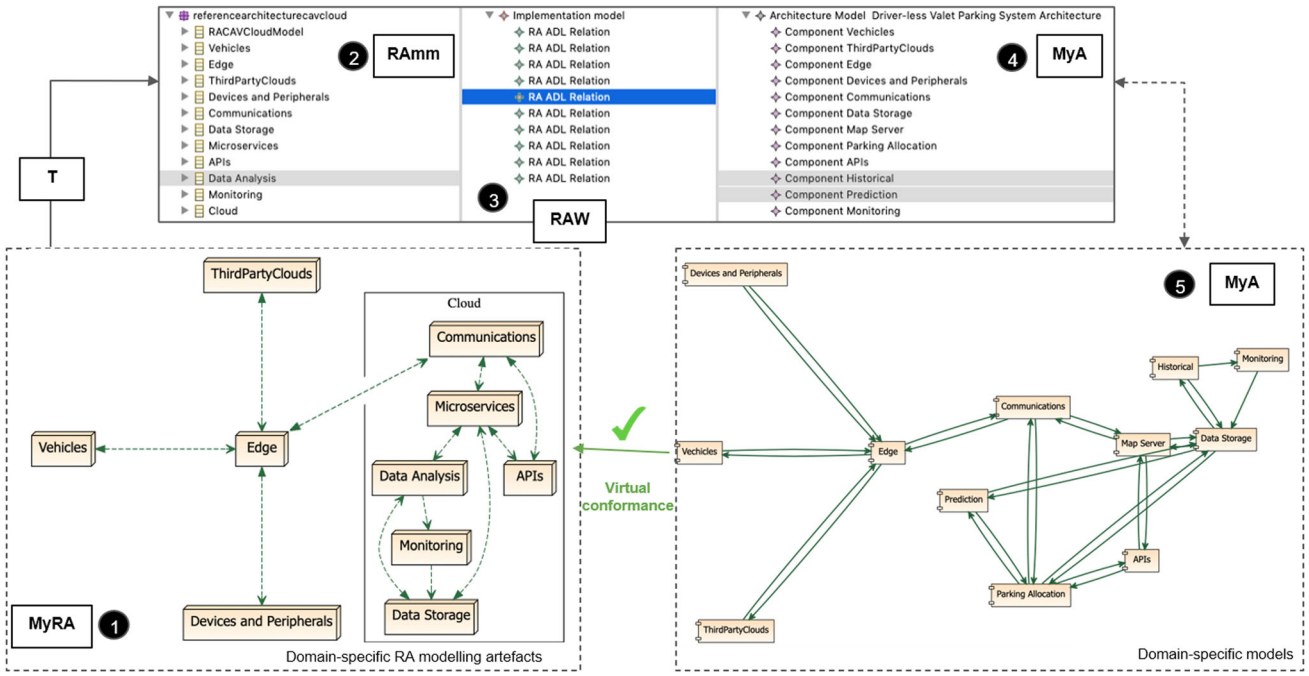


Fig. 13 Application of the proposed approach to the automotive use case in Fig. 12

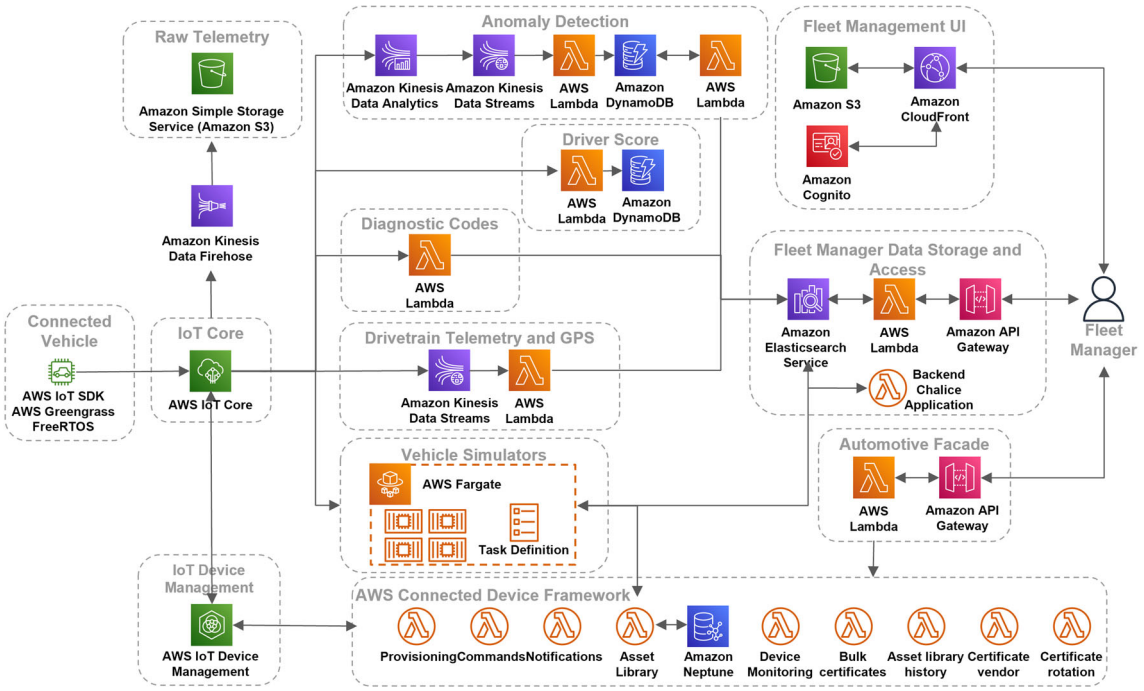


Fig. 14 AWS connected mobility—Reference architecture and architecture

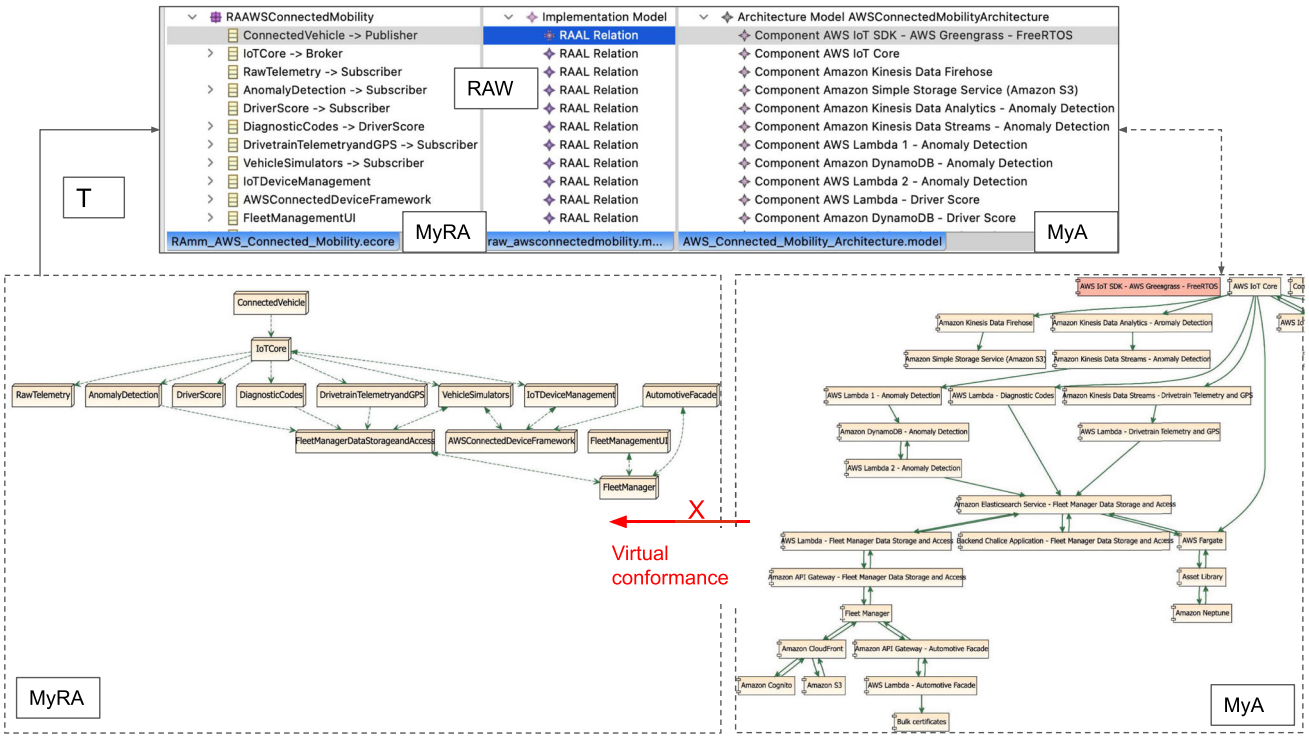


Fig. 15 Application of the proposed approach to the use case in Fig. 14

5.5.2 Internal validity

For all three illustrative examples, we did not directly involve stakeholders working in the field, but rather two authors applied the proposed approach in the context of two different application domains, depending on their technical background and experience. If, on one side, this design decision allowed us to be reasonably confident about the fact that the approach has been applied correctly. On the other hand, we are aware that the two involved researchers might have been biased, given that they also knew the internals of the proposed approach. We mitigated this potential bias by having a third researcher (who did not participate in the definition of the proposed approach) checking the outcomes and modelling artefacts of all three illustrative examples. As already discussed in the previous section, we are already planning an industrial case study where external stakeholders will apply the approach, thus provide further evidence about the applicability and effectiveness of the proposed approach. We attempted to avoid any bias in the definition of the weaving models by considering case studies in which both the RA and the concrete architecture were clearly indicated. This mitigates the construct validity of the tagging operation. Clearly, the modeller defining the RAW might have misinterpreted the architectures, so we included multiple, heterogeneous case studies from different domains and with different complexities. We also highlight that we do not contribute to conceptually define the RAs in this paper since the main contribution relies on the methodology for supporting their definition. All three components of the performed evaluation suffer from the mono-method bias, i.e. using a single type of measures or observations, resulting in a risk of measurement bias [65]. Indeed, we do not compare the proposed approach against any other technique for creating and checking conformance to reference architectures. This design choice was forced because, at the time of writing, there is no generic approach for representing and checking reference architectures comparable to ours.

Finally, our approach intrinsically involves different stakeholders, each with distinct roles, technical background, and knowledge of the considered application domain (e.g. web browsers). The approach might work under the conditions that the involved stakeholders are all sufficiently trained and well aware of the features/steps of the proposed approach. In the context of the performed evaluations, we did not consider this characteristic of the approach and approximated the involved stakeholders by having two different researchers carry out all the approach's steps. The industrial case study planned for future work will mitigate this potential source of bias.

6 Discussion

Representation of RAs and compliance of concrete architectures to RAs are two of the most prominent problems in the field of RAs. In this paper, we propose a model-driven approach for addressing such issues. One concern that can be raised might regard the usefulness of conformance checking between concrete and reference architectures. The need for such a mechanism is, in our opinion, two-fold. On the one hand, misalignment between reference and concrete architectures is one of the main causes of the architectural technical debt (ATD), which is defined as sub-optimal design decisions hindering the evolvability and maintainability of software systems [66]. On the other hand, the virtual conformance checking mechanism presented in this paper is not relevant for reference and concrete architectures only, but for all those situations where specific architectures are derived from more general ones. An example of this is Software Product Lines (SPL), where the architectures of single products must conform to the architecture of the related family [67].

We have demonstrated that our approach is *applicable* to real software systems from *different* application domains. What is more, we have shown that the approach can detect all conformance issues of architectures to a given architectural style and conformance issues between reference and concrete architectures. Indeed, further studies would be needed better to understand the use of the approach in industrial contexts, thus precisely evaluating the return on investment on using the approach, integrating it with existing development processes, organization issues and constraints, and so on.

The proposed approach comprises different artefacts, being metamodels, models, scripts and model transformations. GRAmm and ASV are the metamodel for RAs and the architectural style validation script, respectively. They both contribute to providing means for representing RAs. In particular, GRAmm provides a simple yet effective language, which allows the modelling of any arbitrary RA with concepts close to the architectural domain, e.g. layers, components, connectors, etc. ASV is a collection of scripts that enforce a given architectural style on the modelled RA. It is important to note that this is coherent with the RUP definition of an RA, which is, in its essence, a set of architectural elements and patterns [28]. Besides, ASV brings multiple benefits, including incrementality, reusability, and extensibility. Different ASV could be used for enforcing a given style incrementally: starting with ASVs implementing looser checks and gradually shifting towards ASVs implementing stricter and more rigorous checks. This work describes an ASV for the layered architectural style, but it can be extended/replaced to support any architectural style. Eventually, ASV can be reused among different teams, projects, and even organizations to create a library/repository of ASVs. One of the main limitations of ASV is that its definition might be

cumbersome for the senior software architects as it requires knowledge of constraint languages and metamodelling. On the other hand, ASV is only defined once (per style); hence it represents a one-time effort. The same applies to the RAV script responsible for checking for the virtual conformance of MyA to MyRA being generic and based on the Ecore language. What is more, we have identified additional stakeholders, i.e. the approach maintainers who can support these tasks.

It might be argued that a transformation is not entirely needed as the MyA might be tagged directly to MyRA, without promoting MyRA to a metamodel. Although such a concern is partially true, we believe that there are multiple advantages in promoting MyRA to a metamodel. Model promotion is usually defined as a way to solve the drawbacks of the static types and the explicit dynamic types of approaches using a two-level architecture [47]. In our case, MyRA is used for the conformance checking of MyA, where MyRA first is defined by an architect (dynamic types) and then is used as template to define MyA (static types). For this reason, it seemed natural to promote it to a metamodel. The promotion will then allow checking the conformance of the defined architecture with both the architectural language and the reference architecture. The reference architecture metamodel must be frozen with static types to enable conformance, and virtual conformance checks of the defined architectural model [68]. Graphical editor tools can be used to create concrete syntaxes starting from domain models, i.e. metamodels. This would also enable the use of the elements and concepts of MyRA in the palette of possible built editors for the architect. Just to mention a concrete example, when the architect defines MyRA the components are still dynamic, and only when the transformation is applied and then promoted to a metamodel, an editor could include concepts of the defined MyRA in a toolbox where the architect can select reference architecture elements. This would not be possible and automatable if a model instead of a metamodel is used for the same purpose.

In this respect, we are already working on editors which allow for the creation of architecture models and the double typing on their element to RAmm and ALmm (more details on this can be found in Sect. 8).

ALmm is a simple metamodel for the creation of architecture models. As discussed in Sect. 3, such a metamodel could be replaced with any existing language for modelling software architectures, with small modifications to the process. Hence, it enables extensibility and compliance with legacy technologies. Besides, RAmm makes it easier to write and perform the conformance check performed from RAV. In fact, promoting the MyRA to a metamodel permits to predicate with a generic validation script directly on metamodel construction concepts, instead of navigating the defined MyRA with complex scripts.

Another limitation that could be highlighted is that the weaving model is currently edited manually, meaning that the software architect has to specify the relationships between the MyA and RAmm. Although this is a valid argument, the weaving model can be automatically generated from graphical editors. As above mentioned, we are working towards this by implementing editors able to tag or label MyRA components with MyRA component types. Hence, helping the software architect in defining the weaving relationships graphically. Such editors would also allow for setting models virtually conforming to MyRA, with a live validation process. Currently, the manual editing of the RATagging weaving model RAW is supported by an Epsilon plugin named Modelink[11]. This plugin allows a very intuitive linking operation because the editor is based on the RAW weaving metamodel. For this reason, the architect is forced to correctly link architectural elements by respecting the constraints that the metamodel specifies. This means that the architect can quickly link RA elements to the architectural model with a drag and drop gesture. If the architect knows the domain and the models, this operation takes seconds to identify the right elements in the weaving model. For this reason, the usability of the currently implemented tool may be related to the size of the architectural models. The complexity in this operation corresponds to the browsing operation of the models, in order to correctly identify the model elements to link.

7 Related work

The work in [17] investigates, through an exploratory case study, the benefits and drawbacks perceived by various stakeholders in nine RAs designed by a multinational software consulting company. Among the benefits, they report reduced development costs, improved maintainability and reduced maintenance costs, easier application development and increased productivity of application builders, and alignment with the business need and homogenization of developing and maintaining a portfolio of applications. Among the drawbacks, stakeholders strongly highlight the extra learning curve associated with mastering RAs and limited creativity by giving regulative guidelines to develop applications. The work also summarizes the benefits and drawbacks that they have collected in the literature. The identified benefits are standardization, facilitation, systematic reuse, risk reduction, enhanced quality, interoperability, creation of a knowledge repository, improvement of communication, elaboration of an organization's mission, vision, and strategy, promotion of company-wide best practices, and use of the most novel design solutions. The identified drawbacks are the need for an initial investment, inefficient support for adaptation and

[11] https://www.eclipse.org/epsilon/doc/modelink/.

instantiation, too much abstraction, lack of common interpretation, inadequate documentation, poor quality, and RAs too specific or limiting. Our work contributes to mitigating the drawbacks highlighted in [17]. MORE permits to properly formulate and document RAs while providing a clear interpretation of them. The RA quality will also increase, thus increasing the return on investment and providing support for adaptation and instantiation. More specifically, the work described in this paper proposes a model-driven approach for (i) representing RAs and (ii) checking the conformance of concrete architectures to RAs.

Various studies investigated the use of MDE for RAs. For example, the study in [16] concludes that tools, methods, processes, techniques and technologies used to manipulate metamodels could be used to serve RAs. In [69], the authors propose a model-driven approach for the fast prototyping of web applications. Starting from a RA, an architectural pattern, and a development platform, the approach generates a metamodel of the RA and a graphical editor for manipulating the models, which are instantiated from the generated metamodel. Eventually, the approach translates these models into source code. The approach in [69] is constrained from the use of a specific RA and does not allow users to check the conformance between the concrete and RAs explicitly (although in this case, the conformance is implicit as the concrete architectures, i.e. models, are created employing the editor). The work in [70] proposes an approach for requirements coverage. First, the authors determine a set of architecturally significant requirements. Then, starting from this set of requirements, they define a RA. Eventually, using the RA, they design a decision process-oriented metamodel for designing decision guidance systems. The metamodel is equipped with an editor for modelling design variations and relationships between them. Compared to our approach, the work in [70] leverages the conceptual relation between RAs and metamodels. However, the different phases composing the approach are mostly manual.

Moreover, several ALs during the last decades have been formalized through metamodels, including, e.g. EAST-ADL [71], RCM [72], etc., and an approach has been proposed for using MDE to build new ALs [73].

Similar research to our approach is described in [19], where the authors propose an approach able to generate a domain-specific language based on a metamodel for a predefined RA. The approach can translate the models created using the domain-specific language into source code. Both the approach proposed in this paper and the approach in [19] share the objective of providing automation for the generation of concrete architectures from RAs. However, the approach in [19] is of narrower scope as it only considers a specific RA that is the one used by the Sea Defense Systems Software Team in the ASELSAN project. In our approach, we provide a metamodel for describing arbitrary RAs. Moreover, we provide automatic support for checking the conformance between the concrete and RAs.

In the last decades, several works have been investigating how to check conformance with RAs. Herold et al. describe an approach for conformance checking where the RA and its constraints are formalized as architecture rules enabling automatic conformance checking tool support [74,75]. Compared to our work, the work by Herold et al. is heavily based on the use of architecture rules, which require specific knowledge and are not easy to define. Even when accurately defined, architecture rules tend to have many exceptions to these rules. The authors also acknowledge this in the lessons learned section. Another approach using rules is presented by Weinreich in [76]. In particular, the approach of Weinreich defines RAs using rules consisting of roles and constraints on roles and role relationships. The authors argue that by mapping roles to software architecture elements representation, RAs can be reused. Moreover, as the whole approach is automated, the author argues that it can be applied to different systems.

8 Conclusion and future work

In this work, we have presented MORE, an approach that enables software architects to automatically check the compliance of their architecture description to previously modelled reference architectures. MORE has been implemented with a model-driven approach, providing for a precise definition of reference architectures and an automatic virtual conformance checking mechanism. The definition of reference architectures is entrusted to the proposed Generic Reference Architecture Metamodel. The conformance checking mechanism is entrusted to the Model Promotion Transformation, the Architectural Style Validation Script, the RATagging Weaving metamodel, and the Generated RA Validation Script. To this end, we have introduced the notion of *virtual conformance* inheriting the concept of conformance typical of MDE. To demonstrate the applicability of the proposed approach and its constituent artefacts, we have applied it on two application scenarios inspired by the works in [24], and [26]. The application scenario involved the definition of a reference architecture for web browsers, the definition of concrete architecture for the Mozilla Firefox browser and their virtual conformance checking, as well as a cloud RA for connected and autonomous vehicles for attack surface analysis. We have evaluated the approach and the described tool with a mutation strategy that applies random changes to the modelled browser case study and verifies that the tool can detect invalid RAs or architecture not virtually conform to the declared RA. Eventually, we have discussed the strengths and limitations of the proposed approach. In particular, while we have recognized that the approach might benefit from further improvements, we have shown that it describes a pragmati-

cal and promising solution to the problems of representing reference architectures and checking virtual conformance between concrete and reference architectures. Future work will encompass several research directions. One direction is to extend the approach with architectural style validation scripts for different architectural styles. In time, this might contribute to creating a library, which could serve the entire community. Another direction for future work is to investigate the extension of the AL Metamodel for the representation of concrete architectures. In this context, we are already working on the definition of graphical editors, which can improve the usability of the proposed approach. As we support the possibility of using existing architectural description languages, we will investigate how to integrate these languages within our approach efficiently. Eventually, regardless of whether the AL Metamodel or an existing architectural description language is used, we will explore the possibility of automating the tagging task. To this end, we are planning to integrate MyRA Metamodel and the AL Metamodel (or any other existing language) in editors such that the architect will be able to thoroughly check the virtual conformance during the design of the architecture. Moreover, using the virtual conformance relationship, starting from a RA definition, we can easily generate an automatically exemplary skeleton of concrete architectures, which goes toward automating the instantiation of RA. This seems to be very close to what AWS, for example, refers to as a reference implementation. Eventually, another possible line of future work embraces further validation activities using additional use cases for evaluating different capabilities of the proposed approach, such as scalability and performance.

Funding Open access funding provided by Gran Sasso Science Institute - GSSI within the CRUI-CARE Agreement.

References

1. De Sanctis, M., Spalazzese, R., Trubiani, C.: Qos-based formation of software architectures in the internet of things. In: European Conference on Software Architecture, pp. 178–194. Springer, Cham (2019)
2. Pelliccione, P., Knauss, E., Heldal, R., Magnus Ågren, S., Mallozzi, P., Alminger, A., Borgentun, D.: Automotive architecture framework: the experience of volvo cars. J. Syst. Architect. **77**, 83–100 (2017)
3. Malek, S., Mikic-Rakic, M., Medvidovic, N.: A style-aware architectural middleware for resource-constrained, distributed systems. IEEE Trans. Softw. Eng. **31**(3), 256–272 (2005)
4. Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.L.: Negotiation of service level agreements: an architecture and a search-based approach. In: International Conference on Service-Oriented Computing, pp. 295–306. Springer, New York (2007)
5. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. Addison-Wesley Professional, Boston (2013)
6. Bosch, J.: Software architecture: the next step. In: European Workshop on Software Architecture, pp. 194–199. Springer, Berlin (2004)
7. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall Inc, USA (1996)
8. Guessi, M., de Oliveira, L.B.R., Nakagawa, E.Y.: Representation of reference architectures: a systematic review. In: SEKE, pp. 782–785. (2011)
9. Angelov, S., Grefen, P., Greefhorst, D.: A framework for analysis and design of software reference architectures. Inf. Softw. Technol. **54**(4), 417–431 (2012)
10. Fernández, S.J.M.: Gathering empirical evidence and building a business case for software reference architectures in industry. (2016)
11. Springer Fachmedien Wiesbaden: Autosar has become mature and accepted. ATZextra Worldw. **18**(9), 13–15 (2013)
12. Yimam, D., Fernandez, E.B.: Building compliance and security reference architectures (csra) for cloud systems. In: 2016 IEEE International Conference on Cloud Engineering (IC2E), pp. 147–150. (2016)
13. Eklund, U., Askerdal, Ö., Granholm, J., Alminger, A., Axelsson, J.: Experience of introducing reference architectures in the development of automotive electronic systems. ACM SIGSOFT Softw. Eng. Notes **30**(4), 1–6 (2005)
14. Tokar, J.L.: A comparison of avionics open system architectures. ACM SIGAda Ada Lett. **36**(2), 22–26 (2017)
15. Hochgeschwender, N., Biggs, G., Voos, H.: A reference architecture for deploying component-based robot software and comparison with existing tools. In: 2018 Second IEEE International Conference on Robotic Computing (IRC), pp. 121–128. IEEE (2018)
16. Graciano N., Valdemar V., Garces, L., Guessi, M., de Oliveira, L.B.R., Oquendo, F.: On the equivalence between reference architectures and metamodels. In: 2015 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA), pp. 1–4. IEEE (2015)
17. Martínez-Fernández, S., Ayala, C.P., Franch, X., Marques, H.M.: Benefits and drawbacks of software reference architectures: a case study. Inf. Softw. Technol. **88**, 37–52 (2017)
18. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2012)
19. Turhan, N.K., Oguztüzün, H.: Metamodeling of reference software architecture and automatic code generation. In: Proc. of the 10th European Conference on Software Architecture Workshops, Copenhagen, Denmark, November 28 December 2, p. 2. (2016)
20. Alshuqayran, N.: Static Microservice Architecture Recovery Using Model-Driven Engineering. PhD thesis, University of Brighton. (2020)
21. Alshuqayran, N., Ali, N., Evans, R.: Towards micro service architecture recovery: An empirical study. In: 2018 IEEE International Conference on Software Architecture (ICSA), pp. 47–4709. IEEE (2018)

22. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. Softw. IEEE **20**(5), 42–45 (2003)
23. Bézivin, J.: On the unification power of models. Softw. Syst. Model. **4**(2), 171–188 (2005)
24. Grosskurth, A., Godfrey, M.W.: A reference architecture for web browsers. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 661–664. IEEE (2005)
25. Gómez-Abajo, P., Guerra, E., de Lara, J.: Wodel: a domain-specific language for model mutation. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1968–1973. (2016)
26. Maple, C., Bradbury, M., Le, A.T., Ghirardello, K.: A connected and autonomous vehicle reference architecture for attack surface analysis. Appl. Sci. **9**(23), 5101 (2019)
27. Angelov, S., Grefen, P., Greefhorst, D.: A classification of software reference architectures: analyzing their success and effectiveness. In: 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, pp. 141–150. IEEE (2009)
28. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional, Boston (2004)
29. Behere, S., Törngren, M.: A functional reference architecture for autonomous driving. Inf. Softw. Technol. **73**(C), 136–150 (2016)
30. Pelliccione, P., Knauss, E., Magnus Ågren, S., Heldal, R., Bergenhem, C., Vinel, A., Brunnegård, O.: Beyond connected cars: a systems of systems perspective. Sci. Comput. Program. **191**, 102–414 (2020)
31. Magnusson, A., Laine, L., Lindberg, J.: Rethink ee architecture in automotive to facilitate automation, connectivity, and electro mobility. In: 40th Int. Conference on Software Engineering: Software Engineering in Practice, pp. 65–74. ACM (2018)
32. Bucaioni, A., Pelliccione, P.: Technical architectures for automotive systems. In: 2020 IEEE International Conference on Software Architecture (ICSA), pp. 46–57. (2020)
33. Brogi, A., Carrasco, J., Cubo, J., D'Andria, F., Di Nitto, E., Guerriero, M., Pérez, D., Pimentel, E., Soldani, J.: Seaclouds: an open reference architecture for multi-cloud governance. In: European Conference on Software Architecture, pp. 334–338. Springer, New York (2016)
34. ISO/IEC JTC 1/SC 7 Software and Systems Engineering: Iso/iec/ieee 42010 Systems and Software Engineering – Architecture Description. Standard ISO/IEC TR 29110-1:2016, International Organization for Standardization (2011)
35. Broy, M., Gleirscher, M., Merenda, S., Wild, D., Kluge, P., Krenzer, W.: Toward a holistic and standardized automotive architecture description. Computer **42**(12), 98–101 (2009)
36. Broy, M., Gleirscher, M., Kluge, P., Krenzer, W., Merenda, S., Wild, D.: Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description. Technical Report (2009)
37. Chalé Góngora, H.G., Gaudré, T., Tucci-Piergiovanni, S.: Towards an architectural design framework for automotive systems development. In: Complex Systems Design & Management, pp. 241–258. Springer, New York (2013)
38. Dajsuren, Y.: On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems, PhD thesis. Technische Universiteit Eindhoven (2015)
39. Dajsuren, Y.: Defining Architecture Framework for Automotive Systems, pp. 141–168. Springer, Cham (2019)
40. Fernández-Montes, A., Ortega, J.A., Sánchez-Venzalá, J.I., González-Abril, L.: Software reference architecture for smart environments: perception. Comput. Stand. Interfaces **36**(6), 928–940 (2014)
41. Martínez-Fernández, S., Ayala, C.P., Franch, X., Marques, H.M.: Rearm: a reuse-based economic model for software reference architectures. In: Favaro, J., Morisio, M. (eds.) Safe and Secure Software Reuse, pp. 97–112. Springer, Berlin, Heidelberg (2013)
42. Scott, D.: Gartner hype cycle for real-time infrastructure. Available: https://www.gartner.com/en/documents/2102116/hype-cycle-for-cloud-computing-2012
43. Bézivin, J.: Model driven engineering: an emerging technical space. In: International Summer School on Generative and Transformational Techniques in Software Engineering, pp. 36–64. Springer, New York (2005)
44. Ganesan, D., Keuler, T., Nishimura, Y.: Architecture compliance checking at run-time. Inf. Softw. Technol. **51**(11), 1586 – 1600 (2009). In: Third IEEE International Workshop on Automation of Software Test (AST 2008) Eighth International Conference on Quality Software (QSIC 2008)
45. Monroe, R.T., Kompanek, A., Melton, R.E., Garlan, D.: Architectural styles, design patterns, and objects. IEEE Softw. **14**(1), 43–52 (1997)
46. Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. Softw. Syst. Model. **14**, 02 (2013)
47. De Lara, J., Guerra, E., Sánchez Cuadrado, J.: When and how to use multilevel modelling. ACM Trans. Softw. Eng. Methodol. **24**(2), 1–46 (2014)
48. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education, London (2008)
49. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: On the evolution of ocl for capturing structural constraints in modelling languages. In: Rigorous Methods for Software Construction and Analysis, pp. 204–218. Springer, New York (2009)
50. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The epsilon object language (eol). In: European Conference on Model Driven Architecture-Foundations and Applications, pp. 128–142. Springer, New York (2006)
51. Black, A., Fuad, M.M.: Survey and classification of software technologies for multi-core architecture. In: Proceedings of the ISCA 23rd International Conference on Computer Applications in Industry and Engineering, CAINE 2010, November 8–10 2010, Imperial Palace Hotel, Las Vegas, Nevada, USA, pp. 26–30. (2010)
52. Pruijt, L., Köppe, C., van der Werf, J.M., Brinkkemper, S.: The accuracy of dependency analysis in static architecture compliance checking. Softw. Pract. Exp. **47**(2), 273–309 (2017)
53. Viyović, V., Maksimović, M., Perisić, B.: Sirius: a rapid development of dsm graphical editor. In: IEEE 18th International Conference on Intelligent Engineering Systems INES 2014, pp. 233–238. IEEE (2014)
54. Kolovos, D., García-Domínguez, A., Rose, L.M., Paige, R.F.: Eugenia: towards disciplined and automated development of gmf-based graphical model editors. Softw. Syst. Model. **16**(1), 229–255 (2017)
55. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The epsilon transformation language. In: International Conference on Theory and Practice of Model Transformations, pp. 46–60. Springer, New York (2008)
56. Cuenot, P., Frey, P., Johansson, R., Lönn, H., Papadopoulos, Y., Reiser, M.-O., Sandberg, A., Servat, D., Kolagari, R.T., Törngren, M., Weber, M.: The east-adl architecture description language for automotive embedded software. In: Proceedings of MBEERTS'07, pp. 297–307. Springer, New York (2010)
57. Bucaioni, A., Mubeen, S., Ciccozzi, F., Cicchetti, A., Sjödin, M.: Modelling multi-criticality vehicular software systems: evolution of an industrial component model. Softw. Syst. Model. 2020, To appear, (2020)
58. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C.: The epsilon generation language. In: European Conference on Model Driven Architecture-Foundations and Applications, pp. 1–16. Springer, New York (2008)

59. Kolovos, D., De La Vega, A., Cooper, J.: Efficient generation of graphical model views via lazy model-to-text transformation. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 12–23. (2020)
60. Del Fabro, M.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. In: Proceedings of the 2007 ACM Symposium on Applied Computing, pp. 963–970. (2007)
61. Epsilon: Modelink: Linking Models with Epsilon. http://www.eclipse.org/gmt/epsilon/doc/modelink/. Accessed 20 Mar 2021
62. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. **37**(5), 649–678 (2010)
63. Gómez-Abajo, P., Guerra, E., Lara, J.: Wodel: A Domain-specific Language for Model Mutation, vol. 4, pp. 1968–1973. (2016)
64. Campbell, D.T., Stanley, J.C.: Experimental and Quasi-Experimental Designs for Research. Ravenio Books. (2015)
65. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, New York (2012)
66. Verdecchia, R.: Identifying architectural technical debt in android applications through automated compliance checking. In: 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 35–36. IEEE (2018)
67. Clements, P., Northrop, L.: Software Product Lines. Addison-Wesley Boston, Boston (2002)
68. De Lara, J., Guerra, E., Di Ruscio, D., Di Rocco, J., Cuadrado, J.S., Iovino, L., Pierantonio, A.: Automated reuse of model transformations through typing requirements models. ACM Trans. Softw. Eng. Methodol. (TOSEM) **28**(4), 1–62 (2019)
69. Bernardi, M.L., Di, L., Giuseppe, A., Distante, D.: A model-driven approach for the fast prototyping of web applications. In: 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), pp. 65–74. IEEE (2011)
70. Miksovic, C., Zimmermann, O.: Architecturally significant requirements, reference architecture, and metamodel for knowledge management in information technology services. In: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, pp. 270–279. IEEE (2011)
71. Bucaioni, A., Addazi, L., Cicchetti, A., Ciccozzi, F., Eramo, R., Mubeen, S., Sjödin, M.: Moves: a model-driven methodology for vehicular embedded systems. IEEE Access **6**, 6424–6445 (2018)
72. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., Sjödin, M.: Technology-preserving transition from single-core to multi-core in modelling vehicular systems. In: Springer, Editor, 13th European Conference on Modelling Foundations and Applications. (2017)
73. Di Ruscio, D., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: Developing next generation ADLs through MDE techniques. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, vol. 1, pp. 85–94. (2010)
74. Herold, S., Mair, M., Rausch, A., Schindler, I.: Checking conformance with reference architectures: a case study. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference, pp. 71–80. (2013)
75. Herold, S., Rausch, A.: Complementing model-driven development for the detection of software architecture erosion. In: Atlee, J.M., Baillargeon, R., Chechik, M., France, M., Gray, J., Paige, R.F., Rumpe, B. (eds.) Proceedings of the 5th International Workshop on Modeling in Software Engineering, MiSE 2013, San Francisco, California, USA, May 18-19, 2013, pp. 24–30. IEEE Computer Society (2013)
76. Weinreich, R., Buchgeher, G.: Automatic reference architecture conformance checking for soa-based software systems. In: 2014 IEEE/IFIP Conference on Software Architecture, pp. 95–104 (2014)

Alessio Bucaioni is a software engineer currently working as an assistant professor in computer science at Mälardalen University. His research focuses on several aspects of the development of complex software-intensive systems ranging from software architecture to model-driven engineering. He received his PhD degree from Mälardalen University in 2018. Thereafter, he worked as software engineer and embedded software consultant. During his doctorate and in his research activity, Alessio collaborated with several international companies. Alessio is involved in the organization and program committees for several conferences and is a reviewer for journals in the software engineering domain. Alessio is active in European and national research projects. More information is available at http://www.es.mdh.se/staff/2662-Alessio_Bucaioni.

Amleto Di Salle is currently an Assistant professor at the European University of Rome (Italy) and previously an Assistant professor at the University of L'Aquila (Italy). He received a PhD in computer science from the University of L'Aquila. His research focuses on applying model-based software engineering methods and the methodologies, and formalisms for the analysis, modelling, automatic generation of (micro)service-oriented distributed systems, and technical debt. He has been involved in several European and national research projects.

Ludovico Iovino is Assistant Professor at the GSSI – Gran Sasso Science Institute, L'Aquila—in the Computer Science Scientific Area. His interests include Model-Driven Engineering (MDE), Model Transformations, Metamodel Evolution, code generation and software quality evaluation. Currently, he is working on model-based artefacts and issues related to the metamodel evolution problem. He has been included in program committees of numerous conferences, and in the local organization of the STAF 2015 and iCities 2018 conferences, he organized also the models and evolution workshop at MODELS 2018. He is part of different academic projects related to Model Repositories, model migration tools, and Eclipse Plugins. More information is available at http://www.ludovicoiovino.com.

Ivano Malavolta is Associate professor in the Department of Computer Science and Director of the Network Institute at the Vrije Universiteit Amsterdam. His research interests include data-driven software engineering, with a special emphasis on software architecture, mobile software development, and robotics software. Ivano received his PhD in computer science from the University of L'Aquila, Italy. He is a Member of IEEE, the Association for Computing Machinery, VERSEN, Amsterdam Young Academy, and Amsterdam Data Science.

Patrizio Pelliccione is a Professor in Computer Science at Gran Sasso Science Institute (GSSI). His research topics are mainly in software engineering, software architecture modelling and verification, autonomous systems, and formal methods. He received his PhD in computer science from the University of L'Aquila in Italy. Thereafter, he worked as a senior researcher at the University of Luxembourg in Luxembourg, then assistant professor in the University of L'Aquila in Italy, and then Associate Professor at both Chalmers | University of Gothenburg in Sweden and University of L'Aquila. He has been on the organization and program committees for several top conferences and he is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity, he has collaborated with several companies. More information is available at http://www.patriziopelliccione.com.