Continuous Conformance of Software Architectures

Alessio Bucaioni*, Amleto Di Salle†, Ludovico Iovino†, Leonardo Mariani‡, Patrizio Pelliccione†
* Mälardalen University (Sweden), *alessio.bucaioni@mdu.se*
† Gran Sasso Science Institute (Italy), {*amleto.disalle,ludovico.iovino,patrizio.pelliccione*}*@gssi.it*
‡ University of Milano-Bicocca (Italy), *leonardo.mariani@unimib.it*

***Abstract*—Software architectures are pivotal in the success of software-intensive systems and serve as foundational elements that significantly impact the overall software quality. Reference architectures abstract software elements, define main responsibilities and interactions within a domain, and guide the architectural design of new systems. Using reference architectures offers advantages like enhanced interoperability, cost reduction through reusability, decreased project risks, improved communication, and adherence to best practices. However, these benefits are most pronounced when software architectures align with reference architectures. Deviations from prescribed reference architectures can nullify these benefits. Uncontrolled misalignment can become prohibitively expensive, necessitating costly redevelopments, with maintenance costs reaching up to 90% of development costs. Conformance-checking processes and identifying and resolving violations in the software architecture are essential to mitigate misalignment. To address these challenges, we introduce the concept of continuous conformance that is expressed as a distance function, together with a process supporting it. Continuous conformance quantifies the degree to which a software architecture adheres to a designated reference architecture. The conformance concept enables multi-level, incremental, and non-blocking checking and restoration tasks and allows the check of partial architectures without obstructing the design process. We operationalize this process through an assistive modeling tool to architect an Internet of things-based system.**

***Index Terms*—Reference architecture, conformance, modeling assistant.**

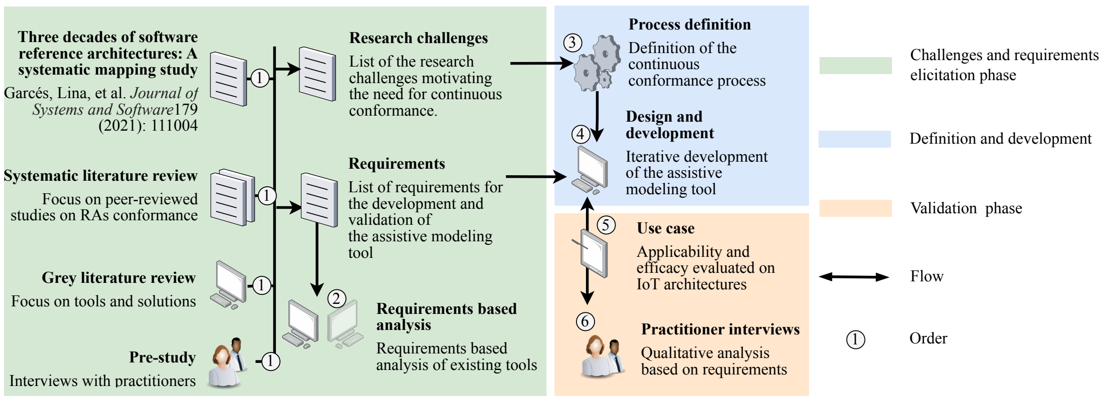
I. Introduction

Software architectures (SAs) are widely recognized as the foundational pillars underpinning the success of software-intensive systems as they play a vital role in determining the overall software quality of these systems [23]. Like many artifacts in the software development process, SAs are not crafted in isolation; rather, they are developed following overarching structures and principles aimed at guiding their development, standardization, and evolution within a specific application domain or organisation [6]. Examples of this include the utilization of, e.g., software product line architectures, architectural frameworks, and Reference Architectures (RAs).

A RA is *"an abstraction of software elements, together with the main responsibilities and interactions of such elements, capturing the essentials of existing software systems in a domain and serving as a guide for the architectural design of new software systems (or versions of them) in the domain"* [13]. In the last decade, researchers and practitioners have created at least 120 RAs, with 51 originating from industry or developed in collaboration with it [13]. Noteworthy examples of RAs include e.g. Autosar in the automotive sector, ARC-IT in cooperative and intelligent transportation, and RAMI 4.0 in Industry 4.0. The use of RAs offers significant advantages, including enhanced interoperability, cost reduction through reusability, decreased project risks, improved communication, and the embrace of best practices [13]. However, these benefits are most evident when SAs align with RAs [17]. Conversely, when SAs deviate significantly from the prescribed RAs, they can nullify the advantages linked to using reference and software architectures [24].

Detecting and minimising misalignment is crucial, particularly for modern software systems that are highly complex and dynamic, leading to continuous evolution of their architectures over time [31]. Left uncontrolled, misalignment can become too expensive to maintain, necessitating costly redevelopments as the maintenance costs for misaligned architectures can reach up to 90% of the development costs [4], [29]. To mitigate and reverse misalignment, it is essential to detect and address it through conformance-checking mechanisms [17], [34]. The results of conformance checking are violations of the software architecture that should be identified and solved to mitigate misalignment. Current approaches for ensuring conformance between software and reference architectures suffer from various limitations including a lack of formalisation, limited automated support, and difficulties with evolution and generalisation. As a result, architects often rely on manual, informal and often subjective checks, which are error-prone, time-consuming and costly [4], [11]. For instance, approaches like the Architecture Compliance Check (ACC) only support manual compliance checking requiring architects to dedicate a substantial amount of time to ensure adherence to the RA incurring up to a 30% overhead on the development time [20].

To address this gap, we propose the concept of *continuous conformance* along with a process. We derived this concept from ore previous definition of conformance [7]. Through a series of elicitation activities, including systematic and grey literature reviews, as well as expert interviews, we gathered a set of challenges and requirements. These inputs led to the refinement of the conformance definition into a distance function, quantifying the degree to which a SA adheres to a designated RA. This redefined conformance concept enables multi-level, incremental, and non-blocking checking and restoration tasks. As a result, these tasks can be executed on partial reference and software architectures without impeding the architectural design process, ensuring continuity. Driven by the elicited requirements, we implemented the continuous conformance process into an assistive modeling tool. This tool supports architects in crafting SAs aligned with specified RAs or more abstract architectures by automatically identifying and rectifying misalignments. To demonstrate the applicability of the continuous conformance process and the accompanying



Three decades of software reference architectures: A systematic mapping study
Garcés, Lina, et al. Journal of Systems and Software 179 (2021): 111004
Systematic literature review
Focus on peer-reviewed studies on RAs conformance
Grey literature review
Focus on tools and solutions
Pre-study
Interviews with practitioners
Research challenges
List of the research challenges motivating the need for continuous conformance.
Requirements
List of requirements for the development and validation of the assistive modeling tool
Requirements based analysis
Requirements based analysis of existing tools
Process definition
Definition of the continuous conformance process
Design and development
Iterative development of the assistive modeling tool
Use case
Applicability and efficacy evaluated on IoT architectures
Practitioner interviews
Qualitative analysis based on requirements
Challenges and requirements elicitation phase
Definition and development
Validation phase
Flow
Order

Fig. 1. Research process

tool, we applied them to a real-world scenario in the Internet of Things (IoT) domain. Additionally, we conducted semi-structured, in-depth validation interviews with various professional profiles to gather qualitative insights into the proposed approach.

The remainder of this paper is organised as follows. Section II provides an overview of the employed research methodology. Section III presents the formalization of continuous conformance and its associated process. Section IV details the proposed assistive modeling tool designed to support continuous conformance. Section V describes the evaluation of the continuous conformance process and the assistive tool. Section VI explores related research in the literature. Finally, Section VII wraps up the paper with concluding remarks and outlines potential directions for future research.

II. Research process

As outlined in Figure 1, our research process uses a combination of complementary research methods, including systematic literature review [18], grey review [14], and expert interviews [26]. This approach is designed to mitigate the limitations associated with a single research method [30]. The research process consists of three primary phases: challenges and requirements elicitation (green box in Figure 1), definition and development (blue box in Figure 1), and validation (orange box in Figure 1).

In the challenges and requirements elicitation phase, we conducted a series of activities aimed at eliciting the set of research challenges motivating the need for a continuous conformance definition and process together with requirements that would guide both the development of the assistive modeling tool supporting such a process and its subsequent validation. We initiated our research by examining the comprehensive study conducted by Garcés et al., which surveyed RAs proposed over the past three decades. Their analysis covered critical aspects, including application domains, context, purpose, perspective, design approach, and maturity. In addition, they identified gaps and outlined potential areas for future work [13].

We complemented the analysis by Garcés et al. with a systematic literature review, which focused on the compliance of RAs. Following the guidelines provided by Kitchenham and Brereton [18], we conducted an automated search across four key scientific databases and software engineering indexing systems: IEEE Xplore Digital Library, ACM Digital Library, SCOPUS, and Web of Science. Striking a balance between rigor and efficiency, we employed a well-constructed search string: "*reference architecture AND conformance*". We meticulously designed the search string to achieve a balance between simplicity and the specificity of the query. The initial search on title and abstract yielded 33 peer-reviewed studies. After eliminating impurities and duplicates, we arrived at a refined set of 19 studies. Following the guidelines by Ali and Petersen [3], we applied a set of selection criteria, which required studies to meet inclusion criteria and avoid exclusion criteria. In this filtering process, we eliminated studies not focusing on RA compliance and obtained a new set of 4 primary studies. To further enhance our review, we performed closed-recursive backward and forward snowballing activities. However, no further studies were selected. Due to space constraints, we can not provide a detailed description of the entire systematic literature review process, but we have made all the artifacts of the review available in our fully public replication package for interested readers [2].

In light of one of the insights from the work by Garcés et al. [13], which highlighted the widespread use of RAs in the industry, we performed a grey review to gather studies specifically focusing on RAs compliance that had not been published as peer-reviewed publications. Our particular focus was on collecting insights from industrial tools and solutions. The grey literature search was conducted using the Google search engine, which constitutes 92.2% of global web searches.[1] The process combined automated and manual activities. It is worth noting that Google automatically removes duplicate entries and conducts optimization, which can lead to variations in results based on factors like time, context, or personalized preferences. [2] The grey review identified the 51 potential tools we filtered, excluding those solutions only providing drawing functionalities. We obtained only one tool, i.e., Amazon AWS composer [1]. As with the literature review, all the artifacts of the grey review are accessible to interested readers in our fully public replication package [2].

The studies from both the literature and grey literature

[1]https://gs.statcounter.com/search-engine-market-share

[2]https://support.google.com/websearch/answer/12412910?hl=en

reviews have been examined using the recommendations by Cruzes et al. [10]. We performed vertical analysis for analysing each study individually and categorising challenges and requirements. The data from the data extraction and synthesis have made available in our fully public replication package for interested readers [2].

Another task in the challenges and requirements elicitation phase involved a pre-study with industry practitioners to gather qualitative insights. We organized online, semi-structured, in-depth elicitation interviews with three professional profiles:

– a system architect working for a Nordic manufacturer among the world leaders in the equipment for construction and related industries,
– the CEO and co-founder of a leading provider of software development environment,
– a cloud solution architect and CTO of a confidential cloud computing provider.

Before each elicitation interview, we provided the practitioner with a preparation sheet that outlined the main interview focus, expected duration, and the privacy and confidentiality measures in place. All interviews were conducted online by the research team and were transcribed into transcripts, with each session lasting between 30 and 45 minutes. During the interviews, we requested practitioners to validate and prioritize the elicited challenges and requirements. We also sought their input on any missing challenges or requirements. No further challenge or requirement was elicited during the pre-study. However, we deleted two requirements being web-based and graphical. The transcripts are accessible to interested readers in our fully public replication package [2].

The outcome of the challenges and requirements elicitation phase was a comprehensive list of challenges and requirements reported in Table I and Table II, respectively. When using the term challenge, we are referring to an open research issue that underscores the significance of a continuous conformance concept and process definition. On the other hand, when using the term requirement, we are specifying the characteristics that a tool supporting such a process should statisfy.

TABLE I
CHALLENGES.

ID	Name	Description	Source
C1	Conformance	Lack of formalisation of conformance	Garcés et al., SLR, PS
C2	Evolution	Definition of a conformance process that can support evolving reference and software architectures	Garcés et al., SLR
C3	Automation	Manual conformance checks are unfeasible	Garcés et al., SLR, PS
C4	Adaptability	The strictness of the conformance checks needs to be adjusted in different phases of the development process	SLR, PS

Each challenge is assigned a unique identifier, a name, a description, and one or more sources. The sources indicate the traceability of the challenge with respect to the study by Garcés et al. [13], the systematic literature review (SLR), the grey review (GR), and the pre-study (PS) with practitioners. Likewise, each requirement is tagged with a unique identifier, a name, a description, one or more sources, and the associated challenge(s). We employed the challenges and requirements identified to analyse the commercial tool discovered during the grey review, assessing their capability to support continuous conformance. The analysis focused on the AWS Composer. While AWS Composer facilitates architecture modeling and automatic generation of related AWS architecture templates, it lacks the functionality to check architecture conformance against a RA. Consequently, architects using this tool must rely on their knowledge to construct the architecture without the guidance of conformance checks. This examination underscored the existing limitations of current commercial solutions in effectively supporting continuous conformance.

TABLE II
REQUIREMENTS

ID	Name	Description	Source	Challenge
R1	Conformance checking	The tool should support conformance checking between architectures	Garcés et al., SLR, PS	C1, C4
R2	Continuity	The tool should support evolving architectures	Garcés et al., SLR	C2, C€
R3	Automation	The tool should support automatic compliance checking and restoration	Garcés et al., SLR, GR, PS	C3
R4	Timeliness	The tool should perform timely automatic compliance checking	SLR	C1, C2
R5	Non-blocking	The compliance checks should not block the architecting process	GR, PS	C1, C2, C4
R6	Knowledge base	The tool should offers data-sets, repositories on reference and software architectures, styles, etc.	SLR, GR, PS	C1, C2, C3
R7	Multi-view	The tool should enable multiple architectural views	GR, PS	C1, C4

In the definition and development phase, we initiated the process with the elicited challenges and requirements as our foundation. Our primary objective was to establish a definition of continuous conformance, outlining its salient properties and a supporting process. Building upon these concepts, we developed an assistive modeling tool designed to streamline the creation of SAs by automatically identifying and correcting misalignment with designated RAs. Inspired by Agile methodologies, we adopted an iterative approach to the development, with each iteration focusing on the most prioritized requirements for that iteration. We present and discuss the definition of continuous conformance, its salient properties and the supporting process in Section III, while Section IV describes the assistive modeling tool.

The validation phase comprises two key activities. In the first activity, we employed a use case from the IoT domain, featuring a RA and a SA for IoT systems. We showcased the applicability of our process by modeling these architectures within the proposed tool, thereby demonstrating the practicality and effectiveness of both the process and the tool in verifying and ensuring compliance between the SA and the RA. Based on the outcomes of this activity, we made refinements to the process and the assistive modeling tool. The second activity focused on gathering qualitative insights from industry practitioners. We conducted online, semi-structured, in-depth validation interviews with five professional profiles, i.e., the three practioners involved in the pre-study plus two additional profiles:

– an embedded system architect working for a manufacturer among the world leaders in the equipment for construction and related industries,
– technology leader working on in-house cloud-based solution for a world-leading provider of transport solutions.

In this activity, our main focus was to gather practitioner feedback on the effectiveness of the process in addressing the identified challenges, as well as on the tool's compliance with the elicited requirements. Each interviewee received an interview preparation sheet outlining the main interview focus, expected duration, and the privacy and confidentiality measures in place. The interviews were held online and audio-recorded, with each session lasting between 30 and 45 minutes. We refined both the process and the assistive modeling tool based on the feedback and assessment provided by the practitioners. All the validation activities are described in Section V.

III. Continuous conformance

The primary contribution of this paper lies in the formulation and delineation of continuous conformance checking along with a process supporting it. As outlined in Section II, the needs for and the requirements of this were discerned through an in-depth examination of the current state of the art via a systematic literature review (SLR), an exploration of existing tools and solutions through a review of grey literature, and a preliminary study involving practitioners. This study included discussions on the concept of continuous conformance, accompanied by a demonstration of its application through the implementation of an assistive modeling tool, detailed in Section IV.

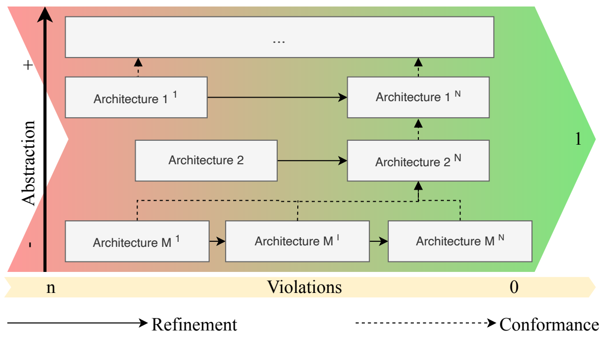


Fig. 2. Multi-level definition of conformance

Figure 2 illustrates the overarching concept of continuous conformance. The vertical axis describes the level of abstraction, encompassing the formulation of architecture descriptions, ranging from the selection and delineation of architectural styles and reference architectures to the specification of software architecture descriptions. The conformance checking operates within the context of a pair of architectures, one exhibiting a higher level of abstraction and the other a more concrete instantiation. These could be, for instance, a style and a RA, RAs at distinct abstraction levels, or a RA paired with a concrete SA [8]. In Section IV, we present diverse scenarios exemplifying its application. The horizontal axis indicates the evolution or refinement of an architecture, RA, style, etc. over time, with an increasing conformance level as the refinement progresses. To introduce the concept of continuous conformance, we initially present some definitions, starting with the definition of architecture.

Definition 1 (Architecture)**.** An architecture $A = (C, CN, CS)$ is a tuple, where:

- $C = (c_1, \dots, c_n)$ is a tuple of components.
- $CN = (cn_a \dots, cn_m)$ is a tuple of connectors.
- $CS = (cs_i, \dots, cs_k)$ is a tuple of constraints, i.e. propositional functions that should be evaluated as true.

We denote A_C as a concrete architecture and A_A as an abstract architecture. The abstract architecture can be, e.g., a style or a RA. A concrete architecture can be, e.g, a software architecture, but also a RA or style in the case of they should conform with another style or RA. Definition 2 introduces the definition of a function to map a concrete architecture to an abstract one.

Definition 2 (Mapping)**.** Given a concrete architecture A_C=$((c_1, \dots, c_n), (cn_1, \cdots, cn_s), CS)$ and an abstract architecture A_A=$((ac_1, \dots, ac_r), (acn_1 \dots, acn_o), ACS)$, the mapping function $map(A_C, A_A) \rightarrow (((c_j, ac_i), \cdots, (c_l, ac_k)), ((cn_e, acn_f), \cdots, (cn_d, acn_v)))$ maps a set of components of A_C to a set of components of A_A and a set of connectors of A_C to a set of connectors of A_A. Each $c_i, 1 \leq i \leq n$ can be mapped to only one $ac_j, 1 \leq j \leq r$.

The designer performs the mapping of components and connectors from A_C to those of A_A during the refinement process. The mapping of connectors can be automated, as described in Section IV. To support incremental and iterative design, we do not mandate that the designer maps every single component of A_C to components of A_A, nor does every component of A_A need to be instantiated in A_C.

In the following, we use $map.c_i$ to denote the abstract component to which a component $c_i \in C$ has been mapped, if any, and $map.ac_i$ to denote the set of concrete components that instantiate the abstract component ac_i, if any.

Definition 3 (Constraints checks)**.** Constraints checks involve two functions for verifying defined constraints in both concrete and abstract architectures. Given a concrete architecture $A_C = (C, CN, CS)$, an abstract architecture $A_A = (AC, ACN, ACS)$, and a map function mapping A_C to A_A, $check_{A_A}(A_C, A_A, map)$ is a function that operates on the architecture A_C. It executes each constraint in ACS that involves components and connectors of A_A that are mapped, through the map function, to components and connectors of A_C and returns the results for each of the executed checks. $check_{A_C}(A_C)$ executes each constraint in CS and returns the results for each of the executed checks.

We can now introduce the concept of connector conformance, as shown in Definition 4, and component conformance, as shown in Definition 5. Before doing so, we introduce two additional instruments that facilitate the formalization. Given a component $c_i \in C$ of a concrete architecture A_C, we denote with $CN_{c_i} = \{cn_1, \ldots, cn_{k1}\}$ the set of connectors of A_C that are connected to c_i and with $ACN_{ac_i} = \{acn_1, \ldots, acn_{k2}\}$ the set of connectors of A_A that are connected to ac_i, where ac_i is the component of A_A mapped to c_i of A_C by a mapping function map.

Definition 4 (Connector Conformance)**.** Given a concrete architecture $A_C = (C, CN, CS)$, an abstract architecture $A_A = (AC, ACN, ACS)$, and a map function mapping A_C to A_A, let $c_1, c_2 \in C$ be two components of A_C. Connector conformance $Conf_{CN}(cn_i, map) \to \{0, 1\}$ is a function that takes as input $cn_i \in CN$, a connector of A_C, and returns 0 if the connector is not legal in A_A, and 1 otherwise. Specifically, let $acn_i \in ACN$ be the connector of A_A to which cn_i is mapped by map, cn_i is not legal in A_A if:

- let $c_j, c_t \in C$ the two components of C to which cn_i is connected, and let $ac_l, ac_p \in AC$ be the two components of AC to which acn_i is connected, the function map does not map c_j and c_t to ac_l and ac_p.

Definition 5 (Component Conformance)**.** Given a concrete architecture $A_C = (C, CN, CS)$, an abstract architecture $A_A = (AC, ACN, ACS)$, and a map function mapping A_C to A_A, the component conformance $ConfC$ is a function defined as follow:

$$Conf_C(c_i, map) = \sum_{j=1}^{|ACN_{ac_i}|} \frac{Conf_{CN}(cn_j, map)}{|CN_{c_i}|} \quad (1)$$

Finally, we can introduce the concept of violations to measure the distance of a concrete architecture from an abstract architecture in terms of violations with respect to a defined mapping, as shown in Definition 6. Additionally, we can introduce the concept of architecture conformance with respect to an abstract architecture, as seen in Definition 7.

Definition 6 (Violation(s))**.** Let $A_C = (C, CN, CS)$, $A_A = (AC, ACN, ACS)$, and map be a concrete architecture, an abstract architecture and a function mapping A_C to A_A, respectively. The number of violations of a A_C with respect to A_A according to the mapping function map is defined as:

$$NV(A_C, A_A, map) = \sum_{i=1}^{|C|} (|CN_{c_i}| - \sum_{j=1}^{|CN_{c_i}|} Conf_{cn_j}(cn_j, map)) \quad (2)$$

Definition 7 (Architecture Conformance)**.** Given a concrete architecture $A_C = (C, CN, CS)$, an abstract architecture $A_A = (AC, ACN, ACS)$, and a map function mapping A_C to A_A, Architecture Conformance $Conf_{A_C}$ is a function defined as follows:

$$Conf_{A_C}(A_C, A_A, map) = \sum_{i=1}^{|C|} \frac{(Conf_C(c_i, map(A_C, A_A)))}{|map(map.c_i)|}$$

In the case of $map.c_i$ is empty, or $map(map.c_i)$ is an empty set, the specific component c_i is not considered by the function.

The function $Conf_{A_C}$ measures the distance to the optimal conformance. In each refinement step, the conformance of A_C to an abstract architecture A_A increases until reaching the value of 1. It is important to highlight that, since the mapping can be partial, we are not forced to check the ideal mapping but only what is declared and actually mapped in a specific iteration.

Considering the challenges identified and described in Table I, the above definitions address the lack of formalisation (C1 in Table I) providing an initial theory for continuous conformance. The choice to define the conformance using a distance function, as opposed to a Boolean function, introduces a more advanced and convincing measure of conformance. Unlike a binary result, a distance function offers a nuanced, quantitative assessment, providing a continuous scale for evaluating conformance [13]. This flexibility is essential in scenarios where strict conformance is not mandatory or desirable throughout the entire development process (C4 in Table I). In the iterative evolution of architectures (C2 in Table I), where changes occur incrementally, a distance function accommodates partial conformance, reflecting the refinement process as described in Figure 2. The quantitative nature of a distance function yields feedback on the extent and specific areas where a concrete architecture deviates from the abstract architecture (violations). These quantitative measures serve as powerful decision support tools, guiding architects in making informed choices during the development life-cycle. The continuous nature of a distance function introduces flexibility, allowing architects to experiment with variations and gradually converge toward the desired architecture. Ultimately, the use of a distance function is of paramount importance for enabling a wide spectrum of automation (C3 in Table I), optimization and search processes, including model repair, discovery, extraction.

IV. An Assistive Modeling Tool Supporting Continuous Conformance

The development of the assistive modeling tool designed to support the continuous conformance process is grounded in model-based techniques, with a focus on the concept of multi-level modeling. This tool empowers software architects to specify various elements, such as architectural styles, RAs, and SAs, using a unified notation with a flexible level of abstraction. This means that each SA can express its conformance to a selected RA, and this process can be iterated across different levels of abstraction employed in architecting software systems. For example, this includes compliance of RAs with architectural styles or SAs with RAs. The notation used is inspired by the work in [7], which is based on a

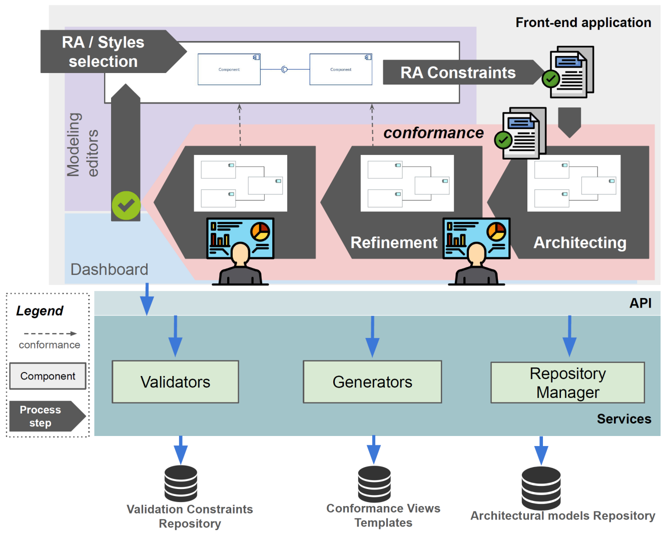


Fig. 3. Overview of the assistive modeling tool

refinement of the component diagram offered by UML [9] and uses the components and connectors view to describe architectures. The choice of such view also aligns with the definition of RA proposed in Section I. Consequently, each artifact in the proposed architecture is model-based or relies on model management operations, resulting in a highly customizable and flexible tool. Figure 3 shows an overview of the assistive modeling tool along with its workflow. The tool is designed as a web-based application, with the back-end services responsible for checking, manipulating, storing, and retrieving models from the central repository. The front-end provides an interface for software architects for continuously architecting software systems while ensuring their conformance.

The front-end comprises modeling editors, which offer architects the ability to visually and textually define architectures. The tool outputs validation results in a console, part of a dashboard referred to as the *conformance view*. This view provides real-time feedback on the current distance to complete conformance, aiding architects in monitoring and addressing conformance issues as they work on the architectures. The front-end interfaces with the back-end services via REST APIs. These APIs facilitate the exchange of architectural models defined in the editors with these services. The chosen technical space for the architectural models is Flexmi [22], a JSON-style notation designed for EMF models, given that the back-end of the application relies on the EMF ecosystem [32]. EMF, a modeling framework based on a structured data model, provides a foundation for modeling both the abstract and concrete syntax of a modeling language, making it suitable for architectural languages. In particular, Flexmi enables the exchange of structured models in a machine-readable format, treated as strings, facilitating compatibility with REST API exchanges. This lightweight format ensures efficient communication between the front-end and back-end services. We identified three main services, that are: *validators*, *generators*, and *repository manager*.

The validators are services designed to take an architecture as input and provide a set of unsatisfied constraints in the event of a failed validation. The assistive tool incorporates two types of validators: one checks the accuracy of the SA at each stage against the selected RA, while the second assesses additional constraints specific to the defined architecture. An excerpt of the validation constraint check of the first type is provided in Listing 1. These validation constraints are expressed in EVL [25], a validation language from the Epsilon language family that enables the formulation of validation rules on models. In this validation rule, the context in which the constraints are evaluated is the component of the architecture, and it applies a guard that selects only the component in which the mapping with the RA is defined. For the components satisfying this condition, the executed check is *checkRelations*, a function that takes the connectors of the selected RA as a parameter.

Listing 1. Excerpt of a validation rule defined on components

```
var raconn = RA!Connector.all;
...
context A!Component {
constraint Relations {
   guard: self.implements.size()>0
    check: self.checkRelations(raconn)
    message: "Violation in ..."
      }
constraint ...{}
}
```

It is crucial to note that the validation depends on the definition of constraints, and from the implementation of the *checkRelations* function, which can be further extended to be more restrictive or relaxed. The validation script that is user-defined from the front-end (second type), is specified using the same language, but it is evaluated at run-time. The combination of these validation results is either a list of unsatisfied constraints or an empty list, indicating successful validation and conformance with the defined architecture. Given the continuous nature of conformance checks, the architecture refinement follows a versioning process, evolving over time. Consequently, the assistive tool fully supports *partial* conformance checks, allowing the *assessment of a slice of the final architecture*. In addition, the tool allows checking the conformance of an SA with a slice of a given RA, by omitting the `implements` definition of a component. The RA validation constraints are stored in a server repository and can be customized and added by the repository manager.

The generators are tasked with producing the conformance view through code generation. The architecture edited on the front-end is processed in conjunction with the selected RA retrieved from the repository to produce a customized view, extendable by specifying additional templates, stored and selected from a dedicated repository. Code generators, expressed in the template-based language EGL from the Epsilon family [28], facilitate this process. One of the main functionalities used by the code generators is the one calculating the *level of conformance* for each component defined in the RA and making this information available through charts and dedicated dashboards. Every time the architectural model is saved (and changed) the views are updated through code generation supplying a real-time feedback of the progress in reaching the conformance.

The auxiliary functions used by the generators are expressed in EOL [21], a model management language of the Epsilon framework. EOL is an abstraction language translating its syntax to Java for executing queries or other model management operations. We reported a small excerpt of the *getConformanceLv* function that returns the value of the conformance calculated over a component of the RA. Indeed, this function is executed on each RA component, by selecting all the components in the architecture implementing it. Then, it builds the conformance level of each component by checking the defined connectors, and in case of failed checks will increase the number of violations and not increase the conformance. The conformance increases for each respected constraint and definition of the RA. Also in this case it is crucial to outline that the script can be customized following the architect's needs or even replaced with a different implementation of conformance. For the sake of demonstration we implemented the conformance as described in Section III.

```
operation RA!Component getConformanceLv( raconn: Set): Real {
var conformanceLv: Sequence;
var components =
M!Component.all.select(c|c.implements.includes(self));
for(c in components){
var connectors =
M!Connector.all.select(con|con.source==c
or con.target==c);
if(connectors.size()>0)
conformanceLv.add(
(connectors.checkConnectors(raconn).sum())
/connectors.size()
);
}
if(components.size()>0)
return conformanceLv.sum()/components.size();
...
}
```

The repository manager takes on the responsibility of retrieving and storing architectures received from the application's front-end. The storage of an architectural model in the architecture repository marks not only its validity in terms of the architectural language, but also ensures adherence to the established validation constraints.

The proposed assistive tool follows a workflow as outlined below. The initial step involves selecting the RA (or architectural style) from the model repository, initiating the validation process for the constraints associated with the chosen RA (details are explained later). The selected RA not only encompasses the structural characteristics of the chosen solution but may also include additional constraints. The architecture definition phase begins with the architect receiving continuous validation assistance and visualizations depicting the distance to achieve conformance. Throughout the refinement of the architecture, continuous checks are performed against the selected RA, promptly updating the dedicated dashboard accordingly. The validation highlights precise errors and warnings detected in the defined architecture up to that point. The refinement phase continues until not only the architecture is fully defined but also conformance is achieved. During the architecture definition, the architect can introduce additional constraints using a dedicated language on the selected RA. These constraints integrate into the defined architecture, competing with those defined by the RA to achieve the final conformance.

We have built a prototype tool available at [2].

V. Evaluation

To assess the continuous conformance process and the accompanying assistive modeling tool, we address the following Research Questions (RQs).

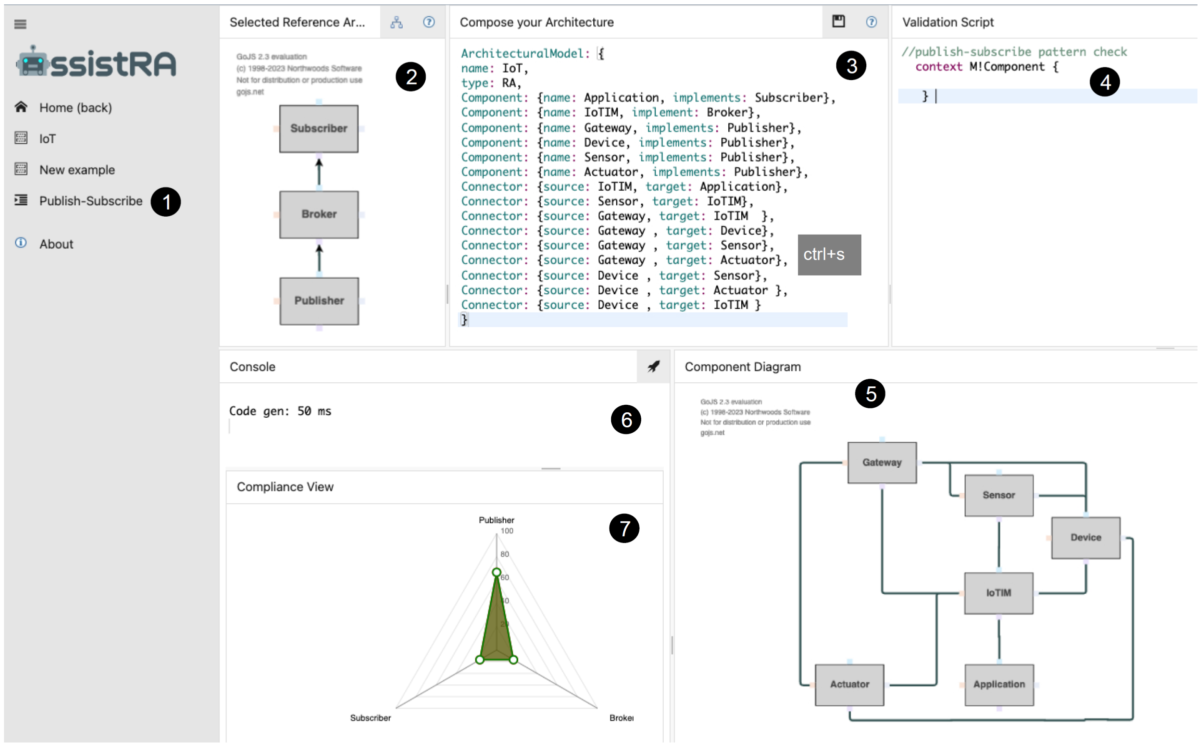
RQ1 – Does the process embodied in the assistive tool address the identified challenges and requirements? By addressing this RQ, our goal is to illustrate how the proposed continuous conformance process embodied in the assistive modeling tool respond to the challenges identified in Table I. We accomplish this by applying the modeling tool to a real-world scenario drawn from previous works in the field [15], [16].

RQ2 – What are the strengths and weaknesses of the proposed concept of continuous conformance, along with the supporting processes and tools? This research question aims to assess the merits and drawbacks of the proposed study. We accomplished this through online, semi-structured, in-depth validation interviews with diverse professional profiles, as outlined in Section II.

A. RQ1 – Does the process embodied in the assistive tool address the identified challenges?

We utilize a scenario from a previous work [15], [16] to demonstrate how the continuous conformance process and supporting tool address the identified challenges. In this scenario, the authors introduce an RA for IoT systems and a set of concrete architectures conforming to the RA. One of the concrete architectures discussed is FIWARE, which we employ to test the process, tool, and workflow outlined earlier. Additionally, we deliberately introduce a violation to validate that the validation process and the generated views effectively highlight the issue. Eventually, we resolve the inconsistency to confirm the capability to address architectural violations.

In the RA, sensors respond to physical stimuli like heat, light, sound, or specific motions to collect information about the physical environment. Actuators, on the other hand, are hardware units that impact the physical environment, translating instructions from connected devices into physical actions, often through electrical impulses. Devices serve as the interface between the physical and digital worlds, connecting to sensors and/or actuators via wired, wireless, or hybrid methods. Drivers encompass processors and storage space. These drivers enable devices to run software, facilitating access to various sensors and actuators. In cases where a device can not directly connect to other systems, it may be linked through a gateway. Gateways play a crucial role in enabling communication across diverse protocols, communication technologies, and payload formats. The IoT RA is designed to be compliant with the Publish-Subscribe architectural style [12]. In this style, communication is expressed between three types of components: the publisher, broker, and subscriber. Figure 4 provides a screenshot of the front-end of the assistive tool. After selecting the style from the repository ❶, a corresponding component and connector view of the



```
ArchitecturalModel: {
name: IoT,
type: RA,
Component: {name: Application, implements: Subscriber},
Component: {name: IoTIM, implement: Broker},
Component: {name: Gateway, implements: Publisher},
Component: {name: Device, implements: Publisher},
Component: {name: Sensor, implements: Publisher},
Component: {name: Actuator, implements: Publisher},
Connector: {source: IoTIM, target: Application},
Connector: {source: Sensor, target: IoTIM},
Connector: {source: Gateway, target: IoTIM  },
Connector: {source: Gateway , target: Device},
Connector: {source: Gateway , target: Sensor},
Connector: {source: Gateway , target: Actuator},
Connector: {source: Device , target: Sensor},
Connector: {source: Device , target: Actuator },
Connector: {source: Device , target: IoTIM }
}
```

```
//publish-subscribe pattern check
  context M!Component {

    } |
```

```
Code gen: 50 ms
```

Fig. 4. Defining a Reference Architecture for IoT

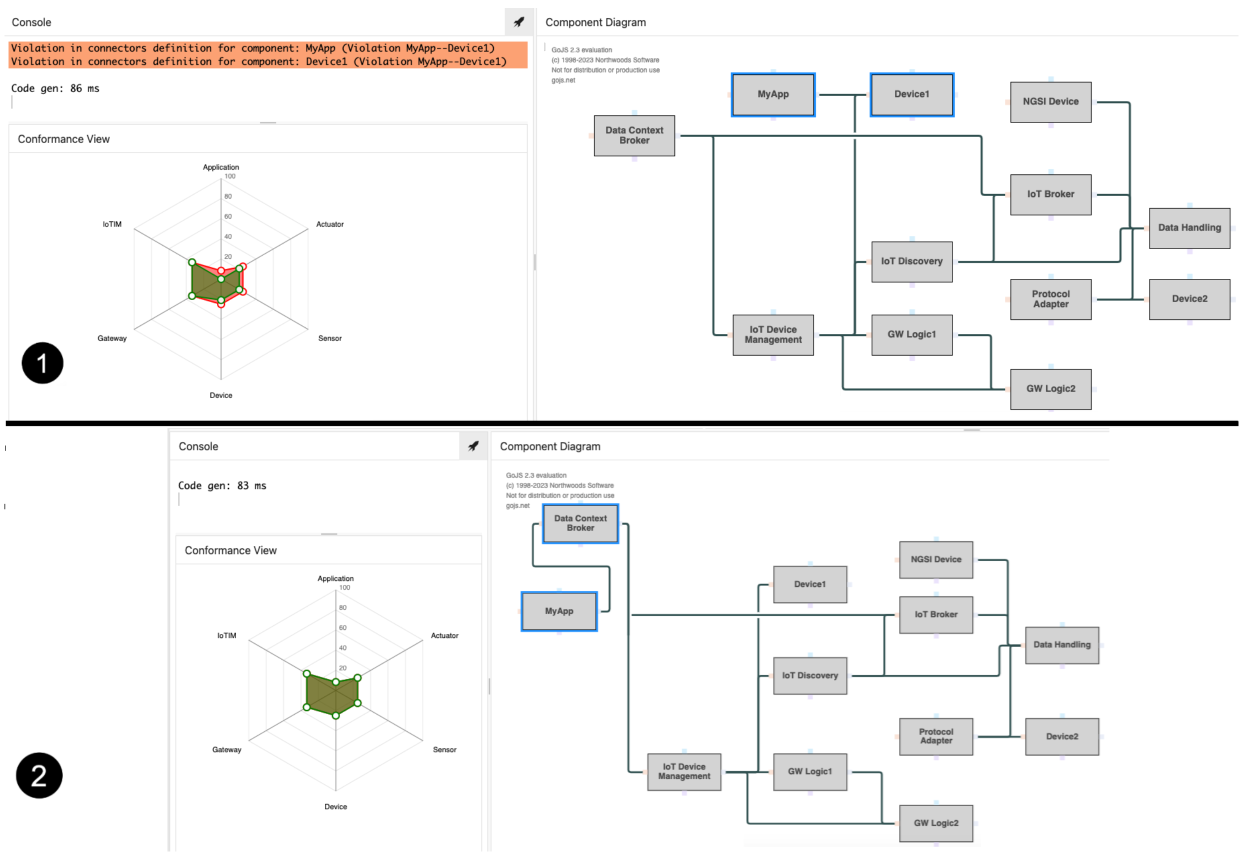
style is displayed 2, providing an immediate visualisation to the architect. The editor 3 is then available to initiate the definition of the RA, while the scripting editor in 4 can be used to define further constraints that will be evaluated at run-time. Each time the architect saves the defined RA 3, the back-end services are triggered, and the defined RA is evaluated against the style and the specified constraints. In Figure 4, the IoT RA is completely defined and the component and connector view of the RA is also generated 5. The console displays the time taken to generate the conformance view 7 and the diagram 5. If the console does not report errors, and the conformance view shows the radar chart all in green, it indicates that conformance has been achieved. The defined architecture can then be submitted to the repository. This final step makes the defined architecture available for selection in the workspace in 1, enabling the IoT RA architecture to be used for defining SA. An example of this is provided in Figure 5 1 where we used the just defined IoT RA for architecting the FIWARE architecture. Figure 5 1 shows that the refinement of the FIWARE architecture led to two possible violations, reported in the console. These violations relate to *Device1* being connected to *MyApp*. This connector is illegally defined as such a connection (between an application and a device) is not expected in the IoT RA, which only considers connectors between application and IoTIM. The conformance view outlines the missing conformance on the RA component application, i.e. the red intersection with the green part. The radar chart is animated by reporting the distance to the conformance of all the implemented instances of the RA components.

Figure 5 2 illustrates the updated FIWARE architecture where the previous violations have been addressed. In the conformance view, the radar no longer includes red areas, and the validation console does not report any existing problems. In particular, a new connection between MyApp and *Data Context Broker* is defined. This connection is a legally declared connector of the RA since all the brokers are typed as IoTIM and can be linked to components of the type application.

This realistic example from the IoT domain serves as a compelling demonstration of the effectiveness of the proposed process and tool, showcasing their utility not only in modeling, but also in the identification of *violations* and *providing guidance for resolution* through the *conformance view*. The experience reported highlights the ability of the defined notion of *continuous conformance* to be *automatically* and *continuously* applied to validate *evolving* architectures, effectively pinpointing issues.

B. RQ2 : What are the strengths and weaknesses of the proposed concept of continuous conformance, along with the supporting processes and tools?

We addressed this question by conducting online, semi-structured, and in-depth evaluation interviews with five professional profiles. Prior to each interview, we provided the participants with a preparation sheet detailing the primary focus of the interview, anticipated duration, and the privacy and confidentiality measures implemented. Throughout the sessions, we presented practitioners with various use cases of the tool. Specifically, we demonstrated the processes for continuous conformance checking, partial conformance checking with a subset of the selected RA, restrictive conformance



```
Violation in connectors definition for component: MyApp (Violation MyApp--Device1)
Violation in connectors definition for component: Device1 (Violation MyApp--Device1)

Code gen: 86 ms
```

```
Code gen: 83 ms
```

Fig. 5. 1: Two conformance violations when architecting the FIWARE architecture. 2: FIWARE architecture conforms to the IoT RA

checking where we adjusted the restrictiveness of checks using additional constraints, and multi-level conformance. Videos illustrating these processes can be accessed in the replication package [2]. The research team conducted all interviews online and meticulously transcribed each session, with duration ranging from 30 to 45 minutes.

All the experts have identified several strengths of the proposed concept of continuous and supporting conformance, along with the supporting processes and tools. Two experts highly appreciated the capability for partial compliance checking, allowing architects to focus on specific subsets of the architectures, which is particularly beneficial in real-world projects where architectures (and standards) evolve continuously. All the experts highlighted the ability to customize and apply more restrictive conformance checks as it demonstrates the tool's adaptability to different domain architectures and stakeholder concerns. In this respect, it is worth to remark that the kind of violations that the tool can detect highly depends on the definition of constraints, and from the implementation of the *checkRelations* function. In the scenario discussed, the violations identified were primarily related to structural mismatches. The automation of conformance checks and the inclusion of a repository for storing developed artifacts was acknowledged for fostering collaboration and building a knowledge base [5]. The multi-view support, encompassing textual and graphical representations, was considered positive, with practitioners praising its flexibility, including the potential for hardware and software coupling views. In addition, the textual representation of validation constraints was noted for the possibility of defining extra constraints beyond the reference architecture. Experts identified additional strengths, noting the tool's value in constructing arguments about adherence to specific, e.g., architectural patterns, particularly in safety-critical systems. An expert highlighted the tool's capacity to offer early feedback in complex scenarios, aiding the impact analysis for architecture changes. Experts underscored the importance of striking a balance between incorporating rich semantics for comprehensive analysis and ensuring usability without excessive effort.

Practitioners identified several weaknesses, too. Firstly, some noted they do not strictly work with conformance, but rather with lighter checks. A practitioner expressed concerns regarding the potential risks associated with specifying constraints that violate the architecture. This vulnerability in handling constraint violations could undermine the tool's overall effectiveness. Additionally, concerns were raised about unintentional architecture breaks during development, highlighting a potential weakness in the tool's ability to prevent unintended deviations. One practitioner highlighted that the process and the tool are better suited for analyzing architectures rather than implementation, which was the main focus of the practitioner's work. Although this limitation could potentially be addressed through code generation features, it is important to note that such features are not fully available at the moment. One practitioner raised concerns about manual input, highlighting that even with the automation of most steps, architects are still required to manually model aspects such as the architecture and constraints. This aspect is noteworthy, as some practitioners find it impractical to allocate a dedicated person for the conformance checking process.

VI. Related Work

Knodel et al. introduced a concept similar to continuous conformance called constructive compliance checking [19]. Their approach aimed to reduce the effort typically required to realign the implementation and its structural overview by proactively preventing such drifts with continuous checks. While Knodel et al. concentrated on ensuring that code adheres to the predefined software architecture, our methodology promotes multi-level conformance, spanning across various layers of the system architecture and implementation. Furthermore, unlike our comprehensive framework, Knodel et al. did not offer a concrete supporting framework but rather discussed the potential advantages of their concept. Pinto and Terra tackled software architectural erosion by proposing a solution for architectural conformance [27]. Their approach adds architectural constraints gradually and checks them during Continuous Integration (CI). Each time code is integrated, it triggers the architectural conformance process. Similarly to Knodel et al., Pinto and Terra only focused on ensuring that code adheres to the predefined software architecture

Several prior investigations have examined methods to guarantee conformance between reference and software architectures, frequently employing MDE approaches. Turhan and Oğuztüzün shared their experience within the Sea Defense Systems (DSS) software team at ASELSAN. They noted that SAs are created in compliance with predefined RAs. Building on this insight, they proposed an approach to facilitate the transition from SAs to code by adopting a model-driven engineering [33]. First, they construct a metamodel for the RA. Subsequently, they employ this metamodel to create a domain-specific language, which, in turn, serves as the basis for designing concrete models, thus giving shape to specific SAs. In the final stage, model transformations automatically transform these models into source code. While our primary focus is not code generation, our approach shares a foundational concept with that of Turhan and Oğuztüzün, which involves representing RAs using metamodels. We leverage this concept to implement the conformance checking that is the pillar of our assistive modeling tool.

Herold et al. conducted a case study exploring the application of model-based conformance checking to ensure compliance with a specific RA [17]. Their study utilized the ArCH approach, which formalizes architectural rules using first-order logic formulas. In ArCH, architectural models are represented as relational structures comprising entities and their relationships, obtained through model transformations. Additionally, model-specific architectural rules are defined in FOL formulas, expressing architecture-specific constraints. ArCH checks architecture conformance by executing these rules as queries to a knowledge representation and reasoning system containing the merged set of relational structures as a fact base. While the study demonstrated the approach effectiveness within their specific use case, it did not assess its suitability for other architectures, particularly those challenging to formalize with FOL formulas. Even in cases where formalization is feasible, it can be overwhelming and may not support evolving architectures. Notably, the ArCH approach did not provide real-time feedback on conformance or recommendations for addressing misalignment when it occurred.

Weinreich et al. proposed an automated approach for verifying the conformance of Service-Oriented Architecture (SOA)-based software systems with their RAs [34]. RAs are defined using rules that specify the roles and constraints on those roles and their relationships through a three-step process. While the proposed approach has benefits, it also relies on some restrictive assumptions. To begin with, it assumes that the RAs and their associated rules are already in place. This may not hold in cases where the software system is being developed from scratch, or a new RA is being introduced or evolved. In addition, the approach assumes that the roles of the RAs can be accurately mapped onto the elements of the software architecture. However, in practice, different mappings may lead to different results, which can impact the effectiveness of the conformance checking process. Eventually, the proposed approach does not guide how to fix the non-conformance identified during the conformance checking process.

VII. Conclusion and future work

This paper addresses the significant challenge of maintaining alignment between software architectures at various abstraction levels, especially as systems evolve. We introduce a conformance relation for continuous assessment, supporting architecture evolution. Our approach is grounded in extensive research, including literature reviews and expert interviews, and is supported by a tool we developed to help architects ensure their designs align with reference architectures consistently. An internet of things example demonstrates the effectiveness of our approach, while the expert feedback highlights the tool's ability to meet identified needs. Future work involves extending our research to diverse domains, exploring applications of the conformance relationship across multiple abstraction levels, and looking into the automatic resolution of detected violations. We envision leveraging Recommender Systems (RS) as advanced assistive modeling tools, seamlessly integrated with visual editors.

Acknowledgment

The work in this paper has been supported by the Swedish Knowledge Foundation (KKS) through the Modev project, by the Excellence in Production Research (XPRES) Framework, by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the iSecure project, by the PNRR MUR project VITALITY (ECS00000041), Spoke 2 ASTRA - "Advanced Space Technologies and Research Alliance", of the PNRR MUR project CHANGES (PE0000020), Spoke 5 "Science and Technologies for Sustainable Diagnostics of Cultural Heritage", and by the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI.

References

[1] "Amazon aws composer," 2023. [Online]. Available: https://aws.amazon.com/application-composer/

[2] "Replication package," 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10798225

[3] N. B. Ali and K. Petersen, "Evaluating strategies for study selection in systematic literature studies," in Procs of ESEM, 2014.

[4] S. Andrews and M. Sheppard, "Software architecture erosion: Impacts, causes, and management." International Journal of Computer Science and Security (IJCSS), vol. 14, no. 2, pp. 82–94, 2020.

[5] P. Avgeriou, "Making decisions - from software architecture theory to practice," 2023. [Online]. Available: https://www.slideshare.net/ParisAvgeriou/making-decisions-from-software-architecture-theory-to-practice

[6] L. Bass, P. Clements, and R. Kazman, Software architecture in practice. Addison-Wesley Professional, 2003.

[7] A. Bucaioni, A. Di Salle, L. Iovino, I. Malavolta, and P. Pelliccione, "Reference architectures modelling and compliance checking," Softw. Syst. Model., vol. 22, no. 3, pp. 891–917, 2023. [Online]. Available: https://doi.org/10.1007/s10270-022-01022-z

[8] A. Bucaioni and P. Pelliccione, "Technical architectures for automotive systems," in 2020 IEEE International Conference on Software Architecture (ICSA). IEEE, 2020.

[9] J. Cheesman and J. Daniels, UML components: a simple process for specifying component-based software. Addison-Wesley Longman Publishing Co., Inc., 2000.

[10] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in Procs of ESEM, 2011.

[11] N. Ernst, R. Kazman, and J. Delange, Technical Debt in Practice: How to Find It and Fix It, 2021.

[12] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe internet architecture," Telecommunication Systems, vol. 51, pp. 233–245, 2012.

[13] L. Garcés, S. Martínez-Fernández, L. Oliveira, P. Valle, C. Ayala, X. Franch, and E. Y. Nakagawa, "Three decades of software reference architectures: A systematic mapping study," Journal of Systems and Software, vol. 179, p. 111004, 2021.

[14] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," Information and Software Technology, vol. 106, pp. 101–121, 2019.

[15] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt, A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences. Springer, 2018, pp. 81–101.

[16] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of iot platform architectures: A field study based on a reference architecture," in Cloudification of the Internet of Things. IEEE, 2016, pp. 1–6.

[17] S. Herold, M. Mair, A. Rausch, and I. Schindler, "Checking conformance with reference architectures: A case study," in 2013 17th IEEE International Enterprise Distributed Object Computing Conference. IEEE, 2013, pp. 71–80.

[18] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," Information and software technology, 2013.

[19] J. Knodel, D. Muthig, and D. Rost, "Constructive architecture compliance checking—an experiment on support by live feedback," in 2008 IEEE International Conference on Software Maintenance. IEEE, 2008, pp. 287–296.

[20] J. Knodel, M. Naab, J. Knodel, and M. Naab, "How to perform the architecture compliance check (acc)?" Pragmatic Evaluation of Software Architectures, pp. 83–94, 2016.

[21] D. S. Kolovos, R. F. Paige, and F. A. Polack, "The epsilon object language (eol)," in European conference on model driven architecture-foundations and applications. Springer, 2006, pp. 128–142.

[22] D. Kolovos and A. de la Vega, "Flexmi: a generic and modular textual syntax for domain-specific modelling," Software and Systems Modeling, vol. 22, no. 4, pp. 1197–1215, 2023.

[23] P. Kruchten, H. Obbink, and J. Stafford, "The past, present, and future for software architecture," IEEE software, vol. 23, no. 2, pp. 22–30, 2006.

[24] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding software architecture erosion: A systematic mapping study," Journal of Software: Evolution and Process, vol. 34, no. 3, p. e2423, 2022.

[25] S. Madani, D. S. Kolovos, and R. F. Paige, "Parallel model validation with epsilon," in Modelling Foundations and Applications: 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings 14. Springer, 2018, pp. 115–131.

[26] J. S. Molléri, K. Petersen, and E. Mendes, "Survey guidelines in software engineering: An annotated review," in Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement, 2016.

[27] A. F. Pinto, R. Terra, E. Guerra, and F. São Sabbas, "Introducing an architectural conformance process in continuous integration." J. Univers. Comput. Sci., vol. 23, no. 8, pp. 769–805, 2017.

[28] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack, "The epsilon generation language," in Model Driven Architecture–Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings 4. Springer, 2008, pp. 1–16.

[29] S. Sarkar, S. Ramachandran, G. S. Kumar, M. K. Iyengar, K. Rangarajan, and S. Sivagnanam, "Modularization of a large-scale business application: A case study," IEEE software, vol. 26, no. 2, pp. 28–35, 2009.

[30] F. Shull, J. Singer, and D. I. Sjøberg, Guide to advanced empirical software engineering, 2007.

[31] E. Soares Palma, E. Yumi Nakagawa, D. M. Barroso Paiva, and M. Istela Cagnin, "Evolving reference architecture description: Guidelines based on iso/iec/ieee 42010," arXiv e-prints, pp. arXiv–2209, 2022.

[32] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, EMF: eclipse modeling framework. Pearson Education, 2008.

[33] N. K. Turhan and H. Oğuztüzün, "Metamodeling of reference software architecture and automatic code generation," in Proccedings of the 10th European Conference on Software Architecture Workshops, 2016, pp. 1–7.

[34] R. Weinreich and G. Buchgeher, "Automatic reference architecture conformance checking for soa-based software systems," in 2014 IEEE/IFIP Conference on Software Architecture. IEEE, 2014, pp. 95–104.