Fundamenta Informaticae (In Press version - Draft)
DOI
IOS Press

Quality-Aware Architectural Model Transformations in Adaptive Mashups User Interfaces

Javier Criado
Applied Computing Group
University of Almería, Spain

David Ameller
GESSI Research Group
Universitat Politècnica de Catalunya, Spain

Nicolás Padilla
Applied Computing Group
University of Almería, Spain

Silverio Martínez-Fernández
Data Engineering
Fraunhofer IESE, Germany

Luis Iribarne
Applied Computing Group
University of Almería, Spain

Andreas Jedlitschka
Data Engineering
Fraunhofer IESE, Germany

Abstract. Mashup user interfaces provides their functionality through the combination of different services. The integration of such services can be solved by using reusable and third-party components. Furthermore, these interfaces must be adapted to user preferences, context changes, user interactions and component availability. Model transformation is a useful mechanism to address this adaptation but normally these operations only focus on the functional requirements. In this sense, quality attributes should be included in the adaptation process to obtain the best adapted mashup user interface. This paper proposes a generic quality-aware transformation process to support the adaptation of software architectures. The transformation process has been applied in ENIA, a geographic information system, by constructing a specific quality model for the adaptation of mashup user interfaces. This model is taken into account for evaluating the different transformation alternatives and choosing the one that maximizes the quality assessments. The approach has been validated by a set of adaptation scenarios that are intended to maximize different quality factors and therefore apply distinct combinations of metrics.

Keywords: quality-driven model transformation, component-based software, architecture metrics, quality model, GQM, QUAMOCO

Address for correspondence: Javier Criado, Applied Computing Group, University of Almería (`javi.criado@ual.es`)

1. Introduction

A particular type of component-based user interfaces (UIs) are those made up of coarse or large grained components, such as widgets or little applications. These UIs within the web domain are called *mashup UIs* because they integrate different services under web technologies [1]. Some examples are offered by Netvibes (`https://www.netvibes.com`), Cyfe (`http://www.cyfe.com`), Dasheroo (`https://www.dasheroo.com`), freeboard (`https://freeboard.io`), Geckoboard (`https://www.geckoboard.com`) and ENIA (`http://acg.ual.es/enia`) applications. This kind of software is usually adapted at run-time depending on user preferences, interactions, or other evolution needs.

Previous studies have shown that model transformation is a good approach to adapt component-based architectures [2]. Existing transformation processes focus on the functionalities of systems, giving less importance to the Quality Attributes (QA). This means that model transformations do not distinguish iso-functional transformation alternatives, although these alternatives fulfill QAs differently. In this sense, if one UI is adapted by only considering its functionalities, such UI may have a less flexible interaction (*e.g.*, complex UI with a greater number of components) or worse maintainability (*e.g.*, costly evolution from introducing unnecessary dependencies among components) than if we consider QAs at run-time. Actually, some QAs (*e.g.*, availability or performance) can only be measured only at run-time since off-line circumstances provide an estimation and not a real value. For example, a user interface of a GIS (Geographic Information System) domain composed by a map and a layer list must be adapted to incorporate a new map with additional information layers. Let us suppose that the system applies a transformation process generating two equal alternatives from a functional point of view but different from the the non-functional perspective. One of them adds a new map with the new information but the other one merges the information offered by the two maps. The former is better in terms of interoperability and accessibility and the latter improves the performance and fault tolerance. Which one should the system choose?

The goal of this paper is to study whether model transformations can be improved by considering QAs at run-time. To this end, we present a QA-aware transformation approach to adapt component-based software systems by measuring the quality of different transformation alternatives. Then, we validate the suitability of such QA-aware transformation approach in four scenarios for the ENIA (ENviromental Information Agent) software. ENIA is a GIS whose UIs are based on coarse-grained components and adapted at run-time depending on user preferences, interactions, system requirements, or other evolution needs [3]. Nevertheless, the approach can be applied to other applications offering their functionality as a mashup or a dashboard.

We propose a set of metrics to measure QAs of ENIA at run-time. We use these metrics to evaluate various alternative architectures (each one obtained by executing a different transformation). As a result, we decide which is the best transformation based on the considered QAs. This article is an extension of the paper originally published in [4]. The contributions of the present research work can be summarized in the next statements:

- A quality-aware transformation has been proposed to support the adaptation of mashup UIs.
- ISO/IEC 25010 quality model has been used to identify the relevant attributes in the ENIA domain from an abstract perspective.

- GQM+Strategies™ has been used for supporting the elicitation of the relevant quality factors and appropriate metrics for accomplishing the ENIA goals.
- QUAMOCO approach has been applied to construct the links between the abstract definitions of the quality factors and the concrete metrics.
- A quality model for the particular domain of ENIA has been built.
- The approach has been validated through four scenarios focused on different quality goals, factors and measures.

This article is structured as follows. Section 2 introduces a background about adapting component-based systems by model transformation. Section 3 reviews the related work present in the literature. Section 4 presents our QA-aware model transformation approach. Sections 5 and 6 exemplify the approach in the ENIA case. Specifically, Section 5 describes the construction of the quality model and the measures in an analysis phase; and next, Section 6 applies these metrics at run-time in different ENIA scenarios. Section 7 discusses the contributions and limitations of the approach. Finally, Section 8 draws the conclusions and proposes some future work.

2. Background

In this section we include the required background for contextualizing the approach of quality-aware architectural transformations at run-time presented in this paper. First, it is important to clarify the particular domain application of our approach, *i.e.*, the adaptation of component based software systems. In the second place, we need to state that this adaptation is carried out by model transformations. Finally, we introduce the specific system which fosters this research work.

Component-based systems are a type of software which facilitates the execution of adaptation and evolution operations. In this sense, well-known mechanisms of Component-Based Software Engineering (CBSE), such as modularization, encapsulation and reuse, favor the development of self-adaptive systems [5]. This software paradigm allows us to manage the components as black-boxes by describing their syntax, semantic, and properties through formal specifications, as in the case of COTS components [6]. Thus, a component can be replaced by other element that matches its specification. Consequently, an architecture can be modified by replacing the parts which need to be adapted.

Model transformation is a common approach to adapt the component-based architecture of software systems [2]. In this context, Model-Based Engineering (MBE) techniques facilitate the development of software architectures, defining them (including the structure, components' specifications, and run-time interaction) by models. Moreover, manipulating architecture models at run-time allows us to generate different alternatives based on the same definition [7]. Depending on the model transformation nature (*e.g.*, vertical, horizontal, endogenous, and exogenous) and within the context of software architectures, it is possible to develop refactoring transformations for obtaining different software alternatives. Our goal is to modify the transformation schema proposed in [2] for generating more than one alternative and consider quality information to select the best transformation.

We addressed this research work focusing on component-based software systems for human-computer interaction. More specifically, we validated our approach by using the scenario of ENIA

UIs, which are used for managing a GIS through coarse-grained components implemented as widgets. Some examples of these components are maps showing geographic layers, visual charts representing datasets, or social widgets enabling the communication with other GIS entities and the community.

ENIA UIs development highlighted the different alternatives that exist when a new architecture is constructed, whether it is determined at design time or it is generated dynamically at run-time. Moreover, such alternatives may be equally valid depending on the quality factors that are taken into account for its construction. For this reason, a quality-aware transformation approach is addressed. Hence, ENIA has been chosen as our test scenario, since the UIs offered by this system are represented and managed as architectures of coarse-grained components (see Figure 1). Each component in ENIA architectures contains the required functionality to perform a task by itself or using its dependencies with other components, *e.g.*, a UI can be formed by a clock, a twitter, a map with rural roads, a map with sea temperature information and a map displaying the water resources. Then, UIs in ENIA are reconfigured at run-time with the aim of adapting their structure to the user interactions, profile preferences, context changes and pro-active system decisions. Since UIs are represented by models conforming to a set of domain-specific languages (DSLs), this adaptation process is based on model-to-model (M2M) transformations (written in ATL transformation language) of component-based architectures (see [2] for further details).

Model transformations in charge of adapting ENIA's UIs are not preset. On the contrary, these operations are dynamically built depending on the initial UI, context information, adaptation rules

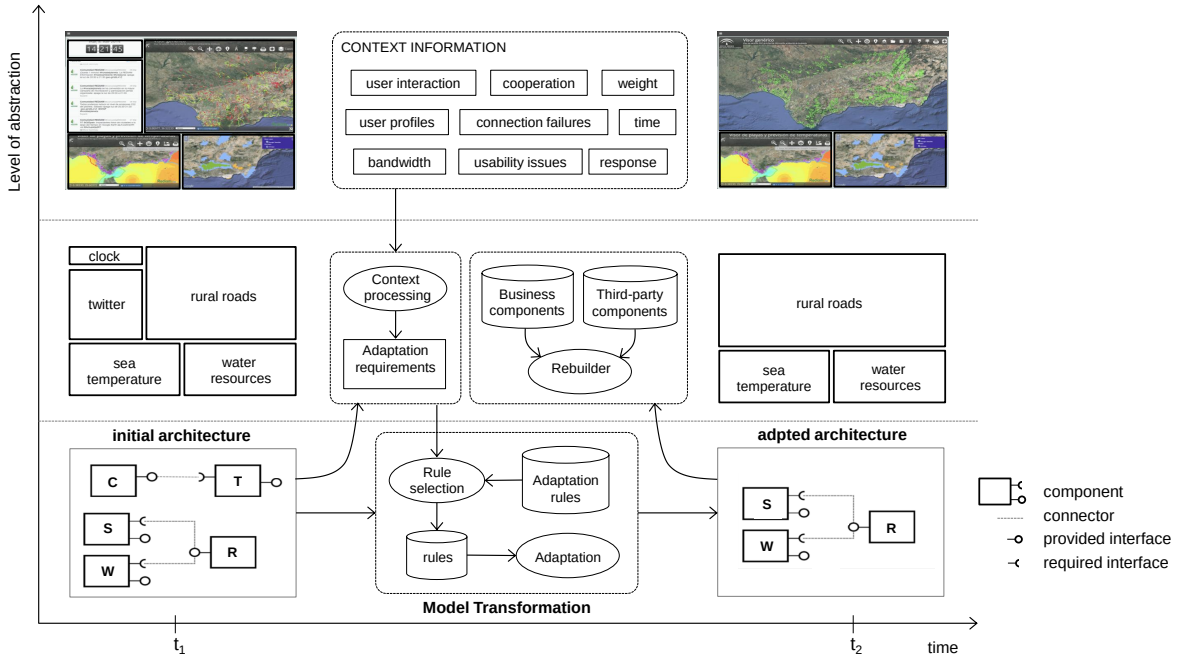

Figure 1. Adaptation by Model Transformation in the context of ENIA

and available components. In this sense, it is possible to build different transformation operations for the same inputs. The transformed models are correct with respect to (a) the source model, (b) the adaptation purposes and (c) the metamodel due to the rule selection and the constraints of the adaptation process. In our previous work [2] we proposed a transformation process that generates a set of possible architectures ensuring the functional resolution of the reference architecture. A tool[1] available as a web application can be used to test this transformation process through an example. In the present paper, we propose a mechanism to rank the iso-functional transformations by incorporating additional quality information thus improving the result obtained from the adaptation process.

3. Related Work

This work covers two fundamental areas of research: Model-Based Engineering and software architecture design. In particular, for these two areas, the contribution of this paper is oriented to the use of quality attributes as the main driver of the adaptations that occur in run-time. In this section we mention works related to our contribution for these identified areas of research.

3.1. Handling Quality Attributes in Model-Based Engineering

Most of the existing model transformation processes focus on the functionalities of systems, giving less importance to the QA, also known as non-functional requirements or *-ilities* [8]. A notable exception are the guidelines for quality-driven model transformations [9], in which quality is introduced early on the design of the transformation process, avoiding quality evaluation as a separate activity once a model has been transformed. A more recent work presented a model transformation framework designed to automate the selection and composition of competing architectural model transformations [10]. However, up to our knowledge, there are few initiatives to select among alternative architectural adaptations at run-time. For instance, a recent European project, SUPERSEDE[2], covers some of these aspects using MBE, but in this case the adaptations are driven by end-used feedback and monitored data rather than the quality attributes. Moreover, there is a lack of empirical evidence of the current situation in the state of the practice regarding the role of quality attributes in the companies adopting MBE approaches (Ameller et al. [11] are working towards such evidence).

Some approaches enable the annotation of model transformations [12] and can be applied for describing QAs in transformation rules. Other proposals extend existing languages with the aim of expressing alternatives and their impact to quality properties at design time [13]. Furthermore, not all QAs share the same importance while adapting or evolving software systems. A recent literature review shows that self-adaptation is primarily used to improve performance, reliability, and flexibility [14]. In this context, an important challenge is to find software architecture metrics that measure quality attributes. The awareness of this problem by the software engineering community is increasing and even dedicated events have been organized [15]. For instance, *dependency structure matrix* metric has been used to measure maintainability [16, 17]. Another examples are the number of components, connections, symbols, and interfaces to measure architectural understandability [18, 19].

[1] Adaptation Tool – `http://acg.ual.es/projects/isoleres/adaptation`
[2] European project SUPERSEDE – `http://www.supersede.eu`

3.2. The Role of Quality Attributes in Software Architecture Design

Quality attributes have an important role in software architecture design. In particular, the architectural decision making is many times based upon the expected quality attributes of the system. This topic has been studied empirically by Ameller et al. [20] in which 12 organizations were surveyed to understand how they use the non-functional requirements to make architectural decisions. There are some works to support software architects in handling quality attributes during the architectural design. For example, Ameller and Franch [21] proposed a process based on an ontology of architectural knowledge to provide alternative architectural solutions depending on the most important quality attributes. The main difference is that in Ameller's work the architectural decisions are planned at design-time, while in the present paper our focus is on architectural adaptations that occurring at run-time. However, there is a recent European project, Q-Rapids[3] [22], working on the integration of quality attributes into the development process from a more general perspective (including both, design-time and run-time).

With regard to component-based software metrics which can be analyzed during the development process, there are different approaches in the literature. Some of them are used for evaluating individual components in isolation, instead of measure the whole system. From other point of view, some proposals are focused on specific technologies, address the measurements of different parts of an architecture, or the granularity level of the architecture elements is not the same [23]. For example, the research work in [24] is focused on CORBA components. The authors of [25] describes metrics related to the interface methods. Concerning the granularity, the approaches in [26, 27, 28] treat the components as large or coarse grained elements which can be managed from the perspective of COTS software. Other approaches not strictly belonging to this perspective but treating the components as black boxes are presented in [29, 30, 31]. Furthermore, there are also proposals of metrics that must be calculated at run-time [32].

4. Quality-Aware Approach for Architectural Transformations

This section presents the proposed QA-aware transformation approach to adapt and evolve software systems by measuring the quality of different transformation alternatives. Such QA-aware transformation approach consists of three steps:

(1) Analyze the relevant QAs and constraints by asking to developers, architects, and experts in the application domain and performing an study of the underlying quality model.

(2) Measuring QAs and constraints at run-time.

(3) Ranking iso-functional alternative software architectures of the model transformation by considering the relevant QAs and constraints at run-time. With this ranking, the software architecture with the best values in architecture metrics is selected.

Next subsections describe the aforementioned steps respectively, which are also depicted in Figure 2. Once the last step is finished and the transformation alternatives have been ranked, the transformation artifact with the best value is executed for adapting the software architecture.

[3]European project Q-Rapids – `http://www.q-rapids.eu`

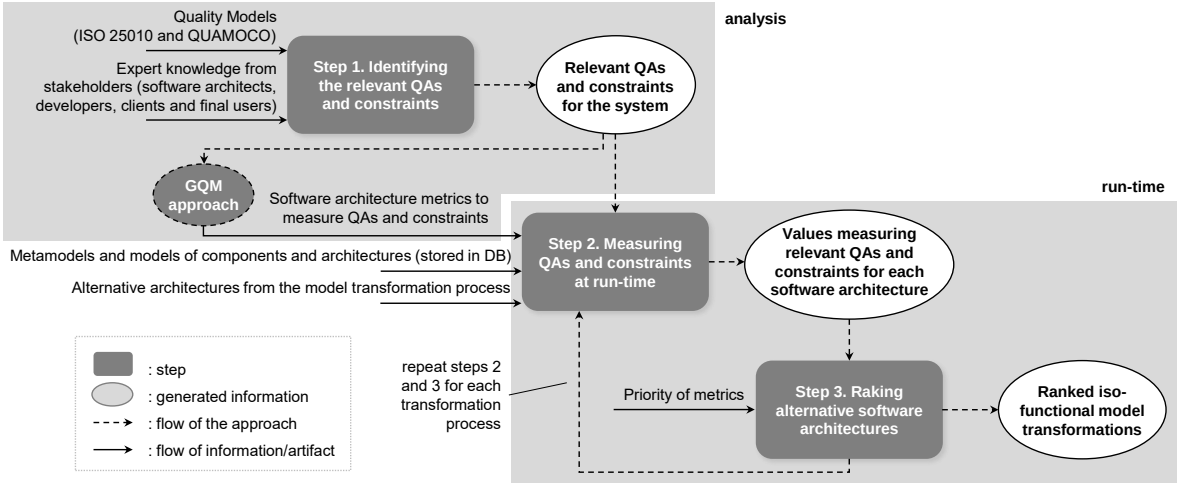

Figure 2. Steps of the QA-based Transformation Approach

4.1. Step 1: Analyze relevant QAs and Constraints

Depending on a system's targeted goals and architecturally-significant QAs (*e.g.*, improve its flexibility, maximize the modifiability, minimize the cost, or optimize the execution performance), architectural design decisions can be oriented in different ways. Therefore, decisions about the construction of software architectures, such as component selection, may differ from each other by considering them. For this reason, the first step of the approach is to gather the architecturally-significant QAs as part of the rationale to make such decisions.

There are several techniques that can be used in this step in different levels of abstraction: from the GQM paradigm [33], to standards proposing QAs that should be studied (e.g., ISO/IEC 25010 standard [34]), to scenario-based approaches to elicit and refine important QAs and architectural decisions (e.g., the Architecture Tradeoff and Analysis Method (ATAM), which has been highly used in the last 15 years [35, 36]), and to recent metrics and data-driven approaches (e.g., Quamoco approach [37]). Due to its capability to construct the traceability from the abstract definition to concrete measurements, we propose to run a workshop in which the stakeholders aims to build a useful quality model following the existing techniques such as the GQM approach and the operationalization techniques of the Quamoco approach to later define concrete metrics from available data sources. This workshop, based on GQM, ISO/IEC 25010 standard, and Quamoco and detailed in Section 5.1, has already been used successfully in other projects[4]. However, other existing techniques such as ATAM could be used in those cases in which operationalizing the relevant QAs by finding concrete measurements is not challenging.

This step requires two inputs: stakeholders who know the system's targeted QAs, and the adequate quality models to help the stakeholders to reason about QAs. With regard to the stakeholders, they are the key elements of the chosen approach to gather the architecturally-significant QAs. They evince

[4]http://blog.iese.fraunhofer.de/competitive-software-improvement/

the relevant adaptation goals in a particular domain. On the other hand, The reason for using existing quality models such as the ISO/IEC standard provides inspiration with the required definitions to identify the relevant attributes related to the quality product (from an abstract procedure perspective).

4.2. Step 2: Measuring QAs and Constraints at Run-Time

Once the set of relevant QAs and constraints are elicited, the result of the first step and the starting point of the second step is to find specific software architecture metrics to measure QAs. As mentioned in the Step 1 and depicted in Figure 2, the metrics have been elicited following the GQM approach. This enables to quantitatively evaluate several alternative software architectures, since the QA satisfaction of these alternative architectures is measured at run-time. Such alternatives are obtained by a dynamic transformation approach [2] which generates different adapted architectures from a common starting architecture (see Figure 3).

The metrics identified in this paper are focused on our particular domain of component-based systems (*i.e.*, ENIA), but they can be adapted according to the needs. Furthermore, some metrics have a generic nature (in this case, generic means applicable to most component-based systems). For example, Table 1 shows a subsets of metrics identified in [4] that are not fixed to a specific domain. Apart from these QA metrics, the mentioned paper also identifies some generic constraints that can be handled in the QA-aware transformation approach. An example of these constrains is the homogenization of components' technology, provider or type. With these constrains it is possible to filter, for example, the software architectures that do not meet a homogenization of 80% in the implementation technology. Other research has also identified generic software metrics as, for example, provided services utilization (PSU), the interface complexity metrics (ICM) and the ratio of component observability (RCO) [23]; or the intra-modular coupling density (ICD), the external relations penalty (ERP) and the groups/components ratio (GCR) [38].

This step requires three inputs: the set of alternative software architectures (generated by the default model transformation process), a set of metrics to measure QAs and constraints, and the specification of the components to feed the metrics at run-time (stored in the component and architecture models [39, 2

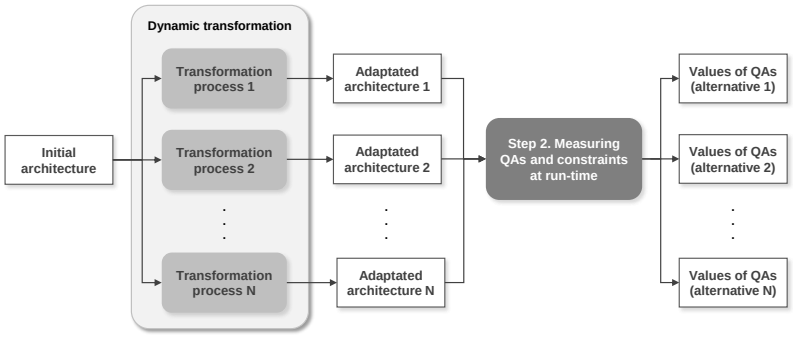

Figure 3. Measuring QAs and constraints at run-time

QA	Metric	Description	Derived	Expression
m	c	Number of components	no	c
	pro	Number of provided interfaces	no	pro
	req	Number of required interfaces	no	req
	$hpro$	Homogenization of provided interfaces	yes	$1 - \sigma^2_{pro}$
	$mdep$	Number of mandatory dependencies	no	$mdep$
	$odep$	Number of optional dependencies	no	$odep$
	dep	Total number of dependencies	yes	$mdep + odep$
	$rmdep$	Ratio of mandatory dependencies	yes	$mdep/dep$
	$rodep$	Ratio of optional dependencies	yes	$odep/dep$
	dsm	Dependency structure matrix	yes	(described in [16])
	pc	Propagation cost	yes	(described in [16])
f	r	Number of resizable components	no	r
	m	The highest c from all alternatives	yes	$max(c_1, ..., c_n)$
	rc	Ratio of components according to m	yes	$rc = c/m$
	rr	Ratio of resizable components	yes	$rr = r/c$
r/a	er	Error rate (and type of error)	no	er
	ec	Error cost	no	ec
p	$extm$	Execution time of a component	no	$extm$
	$rextm$	Ratio of execution time of all components	yes	$\sum(extm_i)/c$
t	$ndiag$	Num. of ops. (in pro) intended for diagnostics	no	$ndiag$
	$ntest$	Num. of ops. (in pro) intended for tests	no	$ntest$
cr	$tsize$	Total size of components (in KB)	no	$tsize$
	$avgsize$	Ratio of components' sizes (in KB)	yes	$tsize/c$
QAs: m: modifiability – f: flexibility – r/a: reliability/analizability – p: performance – t: testability – cr: consumed resources				

Table 1. Example of generic software architecture metrics to measure QAs

metrics must be allocated in these models. Some of this data will be established by the component developer and other data will be initialized and/or updated at run-time during the execution of the software architectures. Relevant metrics and constraints in our approach are described in Section 5. Simple and realistic metrics allow easier adoption in industry [17]. Also, the proposed metrics are just an indicator of a QA, and their improvement must not be seen as a complete satisfaction of any QA. The output of this step is a set of quantitative values measuring the targeted QAs and constraints supporting the selection of the best transformation.

4.3. Step 3: Ordering Alternative Software Architectures

In our approach, we first generate the various possible architectures by applying alternative transformation processes, and then we assess the quality of each architecture. After computing at run-time the corresponding metrics to measure the QAs and constraints, it becomes necessary to rank the alternative software architectures considering the relevant QAs and constraints. Thus, the goal of the third step is to select the "best" software architecture. Consequently, the operation responsible for obtaining this architecture, *i.e.*, the corresponding model transformation, is selected as the best alternative.

This step requires one input: the priority of the architecturally-significant QAs and constraints. This order of importance can be established by system's developers or by users for describing their own priorities. In all cases, it must be specified before the adaptation process starts and could be subsequently modified at run-time to vary this priority. The output is a ranked list of iso-functional model transformation. Finally, the model transformation with the best software architecture is performed. The second and third steps of the approach can be performed at run-time if the number of relevant QAs and alternative architectures to be analyzed is delimited in order to guarantee a proper execution.

5. Applying the Analysis Phase in ENIA

In order to demonstrate the feasibility of the analysis phase of the QA-aware transformation process, this section applies it to ENIA. Next subsections respectively report how the quality model for ENIA has been built, as well as how such model has been operationalized into metrics.

5.1. Building a Quality Model for ENIA

As the Step 1 of the approach indicates, the relevant QAs for the ENIA scenario should be considered for constructing and adapting the UIs. We followed an integrated approach for the creation of a measurable quality model starting from the business' strategic goals, down to quantifiable metrics. The approach is integrated because it shows a workflow and respective moderation methods enabling the use of GQM+Strategies™ [40], QUAMOCO [37], and GQM [33] to build such a quality model, and to visualize the findings. To illustrate the integrating approach, the ENIA case was studied in a workshop with five stakeholders of ENIA.

First, GQM+Strategies™ aligns goals and strategies of an organization across different units through measurement [40]. Besides a clear understanding of what the organization aims at, the use of GQM+Strategies enables communication between different units by getting a common understanding. It helps the development to show their contribution to the higher level key performance indicators. Usually, there are sufficient goals or strategies, which depend on a product quality. During the workshop we identified the following organizational goals behinds software quality:

(a) Scope: the UIs must be used by different kind of people, belonging to different profiles and requiring different sources of information.

(b) Visibility: the UIs must be used by the greatest possible number of people.

(c) Customization: the UIs must allow the modifications and adaptations to the users.

(d) Ease of use: the UIs must be interacted in a friendly way.

(e) Adaptability: the UIs must be adaptable to different platforms and context situations.

(f) Cataloging: the use of the component repository must be optimized by offering the most suitable elements to each user profile and prioritizing the use of the newest components.

(g) Availability: the components and third-party services that are part of the UIs must be available.

(h) Attractiveness: the UIs must be as attractive as possible from the point of view of visual and technical properties.

Second, QUAMOCO solves the problem of traditional software quality models, which provide either abstract quality characteristics or concrete quality measurements, by integrating these two aspects [37]. It provides a generic quality model that needs to be adopted to a company's specific strategic goals. We used the ISO/IEC 25010 [34] as the generic model to match the specific quality goals identified during our workshop sessions. During the workshop, nine product factors were prioritized: (*i*) Response time of the components, (*ii*) Size of the architecture, (*iii*) Degree of promotion of updated components, (*iv*) Stability of architecture, since changes in an architecture indicate that this architecture is not suitable, (*v*) Error rate, (*vi*) Degree of fault recovery, (*vii*) Flexibility, (*viii*) Adaptation to

similar platforms, and (*ix*) Similarity, because if a component has similar components in the repository may be easier to replace it. Figure 4 summarizes the goals and the product factors which were identified during the construction of the quality model for ENIA. It also includes the relationship with the quality factor of the ISO/IEC 25010 standard.

5.2. Measuring Quality in ENIA

GQM provides an approach for goal-oriented measurement. Starting from the goals, questions are derived that have to be answered to know which extend to the goals to be achieved. For quantifying the answers, respective metrics are defined. GQM thus provides the way of how to define a metric and also how to interpret it. We show how we make quality aspects measurable and where to get the data from. Therefore, in our workshop, we ask how to integrate the model, *i.e.*, the measurement, analysis, and derived feedback into their processes.

Focusing on one product factor as an example, we dealt with the flexibility as follows. First, we identified the object, purpose, viewpoint and context around this quality focus by defining a GQM goal: "We must analyze the components and the architectures for the purpose of maximizing the flexibility from the point of view of the user in the context of ENIA UIs interaction". Next, for this quality focus, we proposed a series of question to be answered in the form of metrics. For example, what determines the flexibility of a component? A component is flexible if it is resizable, maximizable and/or if the component allows the grouping. How we can measure the flexibility of an architecture? An indicator is the number of direct and indirect dependencies between components and, in addition, the distribution of the interfaces impacts on the flexibility.

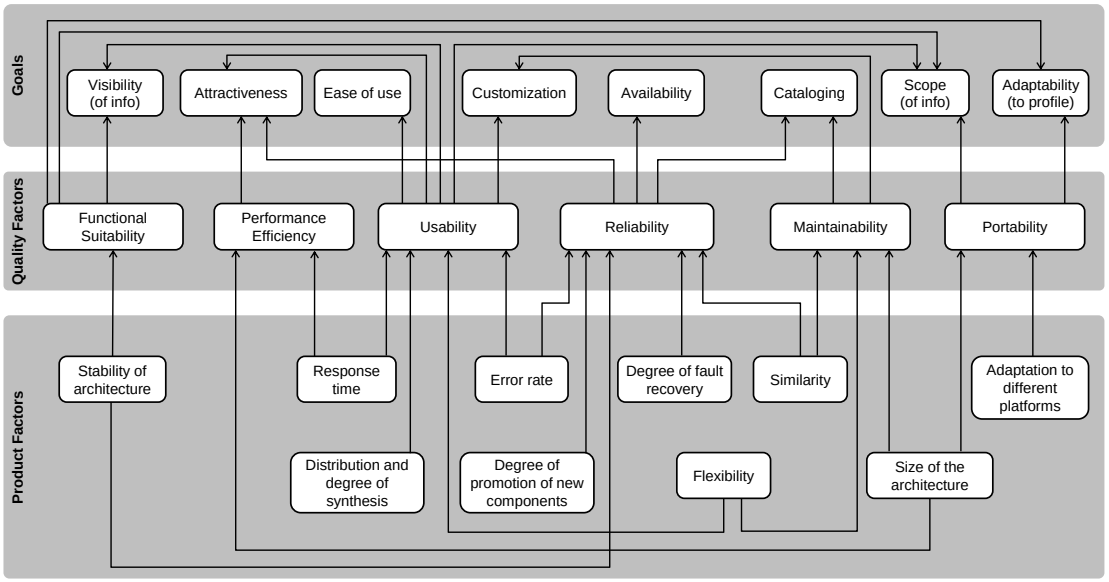

Figure 4. Quality model with related factors

This mechanism to define measurement goals is repeated for each product factor. Tables 2 and 3 describe the metrics that have been identified as valuable and relevant for measuring the quality in ENIA. The first column contains the different product factors (PF) and the second column recalls their relationships with the quality factors (QF) of the ISO/IEC 25010 quality model. The third column establishes the name of the metric, the fourth column identifies the artifact to which it is related, and the fifth and the sixth columns contain the description and the interpretation, respectively.

PF	QF	Metric	Artifact	Description	Calculation / Interpretation
Stability of the architecture	fs, r	Number of changes (nch)	Architecture model	Number of times that an architecture is changed (any modification not related to visual properties)	For each architectural configuration, a counter is stored and incremented each time the architecture is changed. Architectures with a low nch are preferred
		Elapsed time between changes ($etch$)	Architecture model	Average time that an architecture is not changed by the user (in ms)	For each architectural configuration, an attribute stores the total time (tt) that it is deployed and offered to the user. Then, $etch = tt/nch$. Architectures with a high $etch$ are preferred
Size of the architecture	pe, m, p	Number of components (c)	Architecture model	The total number of components present in the architecture	Any type of component (container or not) counts for calculating this metric. Normally, architecture with a high c are preferred, but it depends on other metrics
		Ratio of components (rc)	Architecture model	Ratio of components according to the maximum value from the alternatives	When different alternatives are available, m indicates the highest c from all the architectures. Then, $rc = c/m$. Architectures with a high rc are preferred
		Total size of components ($tsize$)	Architecture model	The addition of the sizes from all the components (in KB)	The size of a component affects negatively the deployment. Architectures with a low $tsize$ are preferred
		Ratio of total size ($rsize$)	Component model	Ratio of a component size with regard to the architecture size	The relative size of a component ($csize$) related to the total size $tsize$. Then, $rsize = csize/tsize$. Components with a low value of $rsize$ are preferred
Response time	pe, u	Time for deployment ($dtime$)	Component model	Average time in which a component is deployed (in ms)	This value is initialized by the developer and is updated each time a component is deployed in a UI. Components with a low value of $dtime$ are preferred
		Execution time of components' methods ($etime$)	Component model	Average time calculated from all the operation execution (in ms)	This value is initialized by the developer and updated each time an operation is executed. Components with a low value of $etime$ are preferred
		Response time of components' methods ($rtime$)	Component model	Average time calculated from all the operation requests (in ms)	This value is initialized by the developer and updated each time an operation is requested and the response is obtained. Components with a low value of $rtime$ are preferred
Distribution and degree of synthesis	u	Number of changes related to the layout ($nlch$)	Architecture model	Number of interactions related to changes in the UI layout	These changes are different from (nch), they are related to visual properties, *e.g.*, width or position in x-axis. Architectures with a low value of ($nlch$) are preferred
Error rate	u, r	Number of timeouts ($ntout$)	Component model	Number of timeouts that a component produces in its execution	This value is updated each time a component produces a timeout error. Components with a low value of $ntout$ are preferred
		Number of times it is not available ($nunav$)	Component model	Number of unavailability errors that a component produces in its execution	This value is updated each time a component produces this kind of error. Components with a low value of $nunav$ are preferred
		Number of console errors (nce)	Component model	Number of console errors (different from $ntout$ and $nunav$)	This value is updated when a component outputs an error in the console. Components with a low value of nce are preferred
Degree of promotion of new components	r	Ratio of new components ($rnewc$)	Architecture model	Ratio of components which have been updated in the last month	It indicates the ratio of new components ($newc$) in relation to c. Then, $rnewc = newc/c$. Architectures with a high value of $rnewc$ are preferred
		Last update ($lupdt$)	Component model	Elapsed time since the last update (in days)	This value is updated when a new version of the component is registered. Components with a low value of $lupdt$ are preferred
		Amount of usage time ($utime$)	Component model	The total time that a component is used by any UI (in s)	This value is updated each time a component ends its deployment in a UI. Components with a high level of $utime$ are preferred
QFs → fs: functional suitability — pe: performance efficiency — u: usability — r: reliability — m: maintainability — p: portability					

Table 2. Metrics to measure QAs identified during the quality workshop for ENIA (part I)

PF	QF	Metric	Artifact	Description	Calculation / Interpretation
Degree of fault recovery	r	Degree of fail-proof (dfp)	Architecture model	Number of components with a total value of error rate under a specific value	The total amount of error rate can be calculated from the addition of $ntout$, $nunav$ and nce. Architectures with a high value of dfp are preferred
		Ratio of low level errors solved (les)	Component model	Ratio of solved error related to console outputs	This value is updated when a developer fixes the component. Components with a low value of les are preferred
		Ratio of medium level errors solved (mes)	Component model	Ratio of solved error related to timeouts	This value is updated when a developer fixes component. Components with a low value of mes are preferred
		Ratio of high level errors solved (hes)	Component model	Ratio of solved error related to unavailability	This value is updated when a developer fixes component. Components with a low value of hes are preferred
Flexibility	u, m	Ratio of resizable components (rr)	Architecture model	Ratio of resizable components in an architecture	This value depends on the number of resizable components (r) and the number of components (c). Then, $rr = r/c$. Architectures with a high rr are preferred
		Ratio of maximizable components (rm)	Architecture model	Ratio of maximizable components in an architecture	This value depends on the number of maximizable components (max) and the number of components (c). Then, $rm = max/c$. Architectures with a high rm are preferred
		Ratio of groupable components (rg)	Architecture model	Ratio of groupable components in an architecture	This value depends on the number of groupable components (g) and the number of components (c). Then, $rg = g/c$. Architectures with a high rg are preferred
		Homogenization of provided interfaces ($hpro$)	Architecture model	Degree of distribution in the number of provided interfaces	This value is calculated from the number of provided interfaces (pro) of each component. Then, $hpro = 1 - \sigma^2_{pro}$. Architectures with a high value of $hpro$ are preferred
		Propagation cost (pc)	Architecture model	Indirect dependencies between components of an architecture	This value depends on the dependency structure matrix of an architecture. The calculation of pc is described in [16]. Architectures with a low value of pc are preferred
Similarity	r, m	Ratio of similarity between two components ($rsim$)	Component model	Degree of proximity between two component definitions	This value is a tuple, $rsim = \{f, nf, p, m\}$, calculated from the matching of two component models distinguishing the functional (f), non-functional (nf), packaging (p) and marketing (m) parts. High values of the tuple terms are preferred
		Number of similar components ($nsim$)	Component model	Number of components with a value of $rsim$ over an specific value	This value indicates the number of 'alternatives' to a component. A component can be considered similar to other when, for example, the value of $rsim$ is greater than of equal to $\{1.0, 0.8, 0.5, 0.0\}$. Components with a high value of $nsim$ are preferred
		Homogenization of providers (hp)	Architecture model	Degree of common providers in an architecture	This value is calculated from the number of components sharing the same provided (sp) and the total (c). Then, $hp = max(sp/c)$. Normally, architectures with a high value of hp are preferred
		Homogenization of types ($htype$)	Architecture model	Degree of common component types in an architecture	This value is calculated from the number of components sharing the same type ($stype$) and the total (c). Then, $htype = max(stype/c)$. Normally, architectures with a high value of $htype$ are preferred
Adaptation to different platforms	p	Number of valid platforms ($vplt$)	Component model	Number of platforms for which a component is adapted	This value is established by the developer. Components with a high value of $vplt$ are preferred
		Degree of adaptability to other platforms ($dadp$)	Architecture model	Ratio of components which are valid for at least two platforms	This value is calculated from the value of $vplt$ for each component and the number of components (c). Architectures with a high value of $dadp$ are preferred
QFs → fs: functional suitability — pe: performance efficiency — u: usability — r: reliability — m: maintainability — p: portability					

Table 3. Metrics to measure QAs identified during the quality workshop for ENIA (part II)

In the next section, we present four scenarios in the context of ENIA. In the first scenario, we analyzed three metrics related to functional suitability, usability and portability to maximize the scope of the UIs. The second scenario is intended to maximize the attractiveness by means of three metrics related to performance efficiency, usability and reliability. The third scenario uses four metrics to maximize the customization, and finally, in the last scenario, we applied three constraints for maximizing the efficiency of the catalog. Both scenarios share components related to the GIS domain,

for example, a map (M) for displaying geographical information layers, a component showing the messages of a Twitter account (T), or a layer list for selecting the information to be displayed on the map (LL). Otherwise, the number of the component name represents different alternatives of the same component type, *i.e.*, *M1*, *M2* and *M3* are different alternatives of the map component type M.

6. Using the Quality Model and Measures in the Run-Time Phase

At the end, the model has to be used at run-time to support our transformation of architectures, intended for adapting our ENIA UIs. As mentioned in Section 4, the run-time stage of our approach calculates the values for the selected metrics and ranks the alternatives of architectures with the aim of ranking the corresponding transformations and select the best one (steps 2 and 3 of Figure 2). In order to illustrate these steps, this section presents four scenarios for applying the quality model and the metrics in ENIA. Each scenario focuses on reaching different goals and, consequently, different kind of metrics. Moreover, a example subset from all the possible metrics presented in Tables 2 and 3 have been applied, and each scenario describes the criteria to prioritize the metrics.

6.1. Scenario 1: maximizing the scope

The first scenario starts from a UI made up of three components (see Figure 5). The first component is a map providing the functionalities of showing geographic information (provided interface M) and showing the corresponding legend (provided interface L). The second component is a configurator of the legend settings (*e.g.*, colors or show/hide text labels) and the third component is a Twitter widget showing relevant information for the REDIAM users. Then, the UI requires to be adapted by removing the functionality of the legend configurator. The motivation of the adaptation can be a pro-
the s

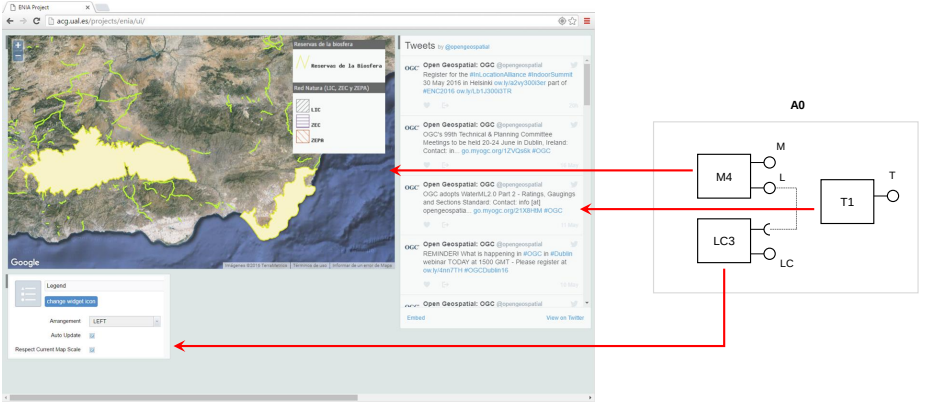

Figure 5. Initial UI of scenario 1 and its architecture

This adaptation process can be simple from the functional point of view, because the transformation could be accomplished by removing the $LC3$ component. Nevertheless, our approach is intended to take into account the rest of non-functional features related to the new version of the architecture. In this case, our goal is to maximize the *scope* and, as depicted in Figure 4, the quality factors related to this goal are: functional suitability, usability and portability. Furthermore, we have chosen three product factors having impact on this quality factors: stability of the architecture, distribution and degree of synthesis and adaptation to different platforms. In particular, we have chosen the following metrics of Tables 2 and 3:

(a) $etch$ – The elapsed time between changes must be maximized because the more time an architecture is not changed, the more stability can be assumed. As a consequence, the stability of the architecture is greater, impacting positively on the functional suitability and improving the possible scope by offering a more stable UI.

(b) $nlch$ – The number of changes related to the UI layout must be minimized because a high value of this measure indicates that the distribution and degree of synthesis are not adequate. Therefore, it affects negatively the usability and it can reduce the number of final user interacting with the UI (*i.e.*, the scope).

(c) $dadp$ – The degree of adaptability to other platforms must be maximized since it can have a positive influence to the scope due to the improvement of the portability of the UI. The more platforms that can correctly deploy the architecture, the greater scope of that UI.

Figure 6 shows the three transformations alternatives that are generated from the initial UI described in the architecture $A0$. Such alternatives are specific for this scenario of removing the legend configurator. The notation of the architectures is depicted in the key inside the figure and the semantics of this notation is the common use of software architectures [41]. The first alternative $A11$ replaces the map component $M4$ by a new one ($M1$) which have no legend functionality. As a consequence, a new legend is inserted ($L1$). The alternative $A12$ replace the map component too but, in this case, the new map ($M2$) and the new legend ($L2$) require to be wrapped by a container ($C1$). The third architecture $A13$ is the alternative of removing the component $LC2$ from the UI.

Then, the selected metrics are applied to be able to rank the model transformation processes ($MT1$, $MT2$ and $MT3$). For example, the value of $etch$ for $A11$ is 10 seconds because the total time without changes (tt) is 20 seconds and the number of changes (nch) is 2. The values of the three alternatives are normalized to combine their values. In the case of $etch$, the normalized value is calculated by dividing each value by the maximum of the three alternatives (because it must be maximized). Therefore, the normalized value of $etch$ for $A11$ is 0.77 resulting from $(tt/nch)/maxetch = (20/2)/12.86$. Regarding the metric $nlch$, the normalized value is obtained by the expression $1 - nlch/maxnlch$, where $maxnlch$ represents the maximum of the three alternatives (because it must be minimized). For example, the value of the normalized value of $nlch$ for $A11$ is 0.67 obtained from $10/30$. The value of $dadp$ for $A11$ is 0.33 resulting from dividing the total of components that are valid for at least two platforms by the number of components ($1/3$). The best transformation alternative is selected by ranking the average of the three normalized values (because the weight for each metric is the same). It is 0.59, 0.33, and 0.67, corresponding to $MT1$, $MT2$

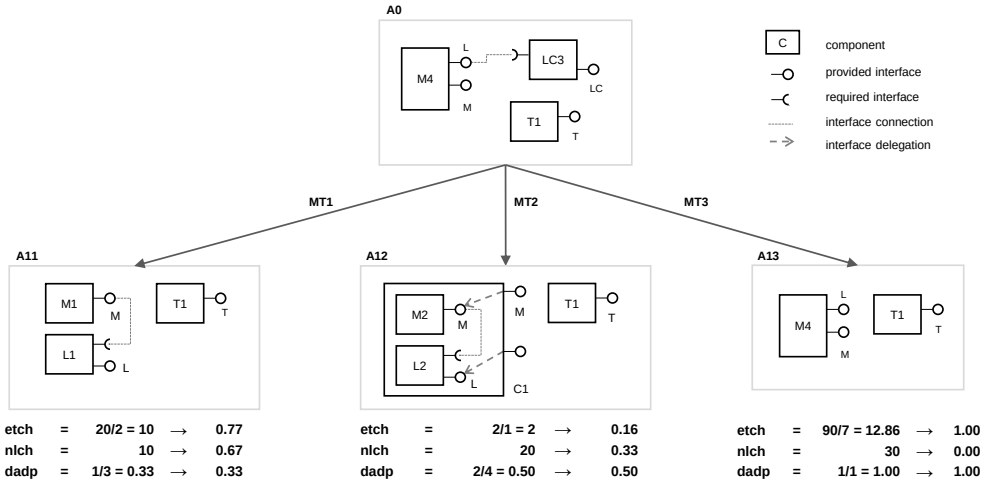

Figure 6. Transformation alternatives of scenario 1

and $MT3$, respectively. Therefore, $MT3$ is executed and the UI related to the architecture $A13$ is deployed and shown to the user.

6.2. Scenario 2: maximizing the attractiveness

Starting from the user interface obtained in the scenario 1, the second example shows the usage of our approach in the case that a list of the geographical information layers is added to the UI. For this scenario, we are going to focus on the goal of maximize the *attractiveness* of the interface. Thus, the related quality factors are performance efficiency, usability and reliability; and we select a subset of products factors related to them and to the goal: response time, error rate and degree of promotion of new components. Next, we choose some metrics which have been identified in the quality model:

(a) $dtime$ – The deployment of the components must be minimized since it affects negatively the performance efficiency and, consequently, the attractiveness of a UI.

(b) $nunav$ – A high value for the number of times that a component cause an error of unavailability has a negative impact on the usability and reliability, and derived from that, the UI would be less attractive for users. Therefore, the $nunav$ must be minimized.

(c) $lupdt$ – The last update performed in the components must be minimized because the presence of new components affect positively the reliability and the goal of attractiveness, if we assume that updates always improve some feature of a component.

The four alternatives of architectures to resolve the required functionality starting from $A13$ are shown in Figure 7. The alternative $A21$ replaces the map by $M1$ and provided the legend (L) and layer list (LL) functionalities in a new component $LL1$. The architecture $A22$ is similar to $A21$ but it provides the L and LL functionalities in two separated components ($L1$ and $LL2$). The third

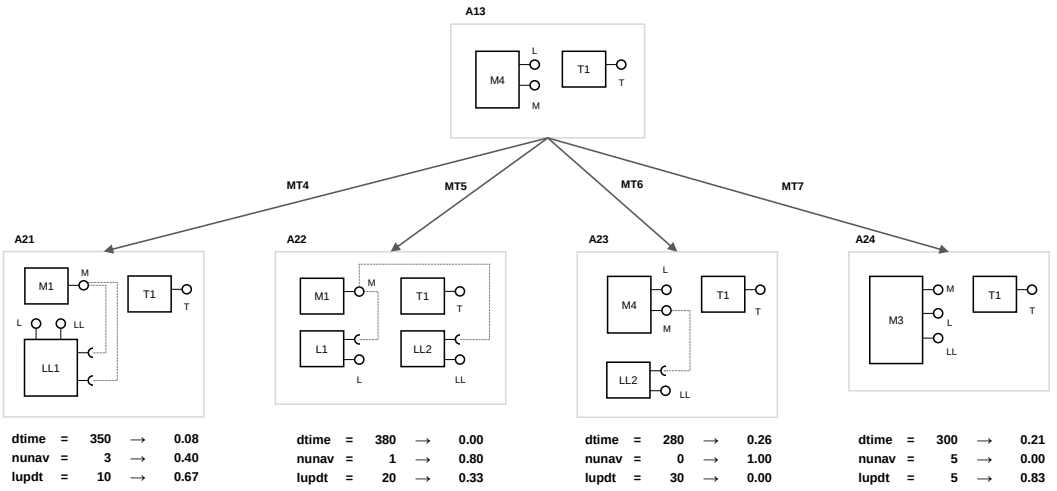

Figure 7. Transformation alternatives of scenario 2

possible architecture $A23$ is the simplest alternative because it solves the functional requirements by incorporating a new component $LL2$. The last alternative gathers the M, L and LL functionalities in one component $M3$.

When the mentioned metrics are calculated for the four alternative architectures, we obtain the values summarized in Figure 7. The first value for each metric shows the stored data for each component. The second value shows the normalized data to operate with the three metrics in the same range. In this case, because all the value must be minimized to maximize the attractiveness, the operation to normalize the values is similar: $1 - val/maxval$; where val corresponds to the value of the metric and $maxval$ represents the maximum value for this metric from all the alternatives.

If we give the same importance to the three metrics and perform their average, the results are 0.383, 0.377, 0.420 and 0.347, corresponding to $MT4$, $MT5$, $MT6$ and $MT7$, respectively. But, in this scenario, the weights of the three product factors are 10% for the response time, 50% for the error rate and 40% for the degree of promotion of new components. In this case, there is only one metric related to each product factor and therefore, the eeight of each metric with respect to its product factor is 100%. Based on the QUAMOCO approach, the metrics are first calculated, then normalized and finally the weights are applied for aggregation. Figure 8 shows the approach to calculate the value of attractiveness for the architecture $A21$. Consequently, the values of attractiveness calculated for the four transformations are 0.476 ($MT4$), 0.532 ($MT5$), 0.526 ($MT6$) and 0.353 ($MT7$). Hence, the best transformation is $MT5$ and the architecture $A22$ is deployed as shown to the user (see Figure 9).

6.3. Scenario 3: maximizing the customization

The third example to describe our approach and validate the proposed quality model is focused on the goal of maximizing the customization of the UI. In this example, the transformation process is aimed to incorporate a new functionality for managing the session of the interface and, moreover, the

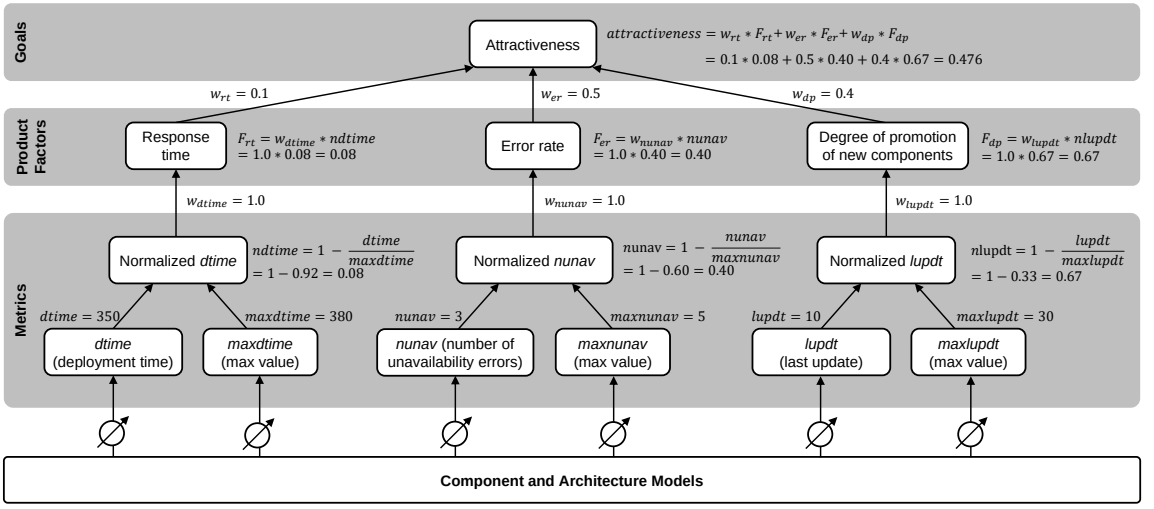

Figure 8. Applying QUAMOCO [37] to structure the measures and perform quality assessments

session information must be used by the social component (Twitter) to show the corresponding profile information. Due to the selected goal, we must deal with the usability and maintainability quality factors. Specifically, we have selected the size of architecture and flexibility product factors. Then, we have focused on a subset of metrics among all the available ones of the constructed quality model:

(a) rc – The ratio of components must be maximized because the more pieces constitute a UI, the more reconfiguration and modification operations can be performed on its architecture.

(b) rr – The ratio of resizable components must be maximized since this property favors the flexibility of a U

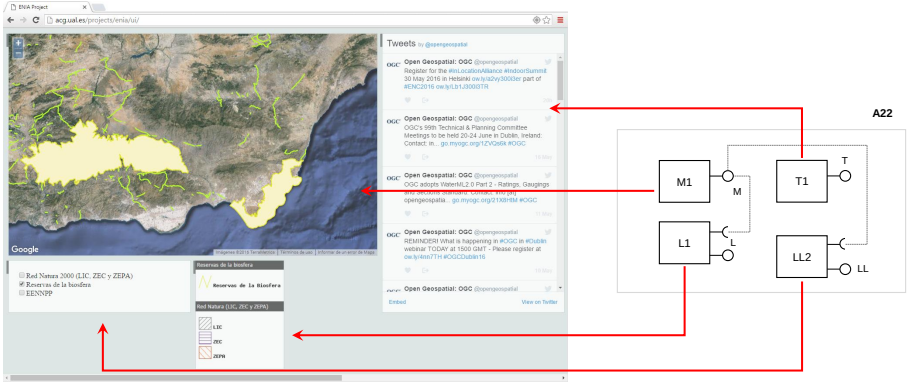

Figure 9. Resulting UI of scenario 2 and initial UI of scenario 3

(c) $hpro$ – The homogenization of the distribution of provided interfaces must be maximized because this property avoids the imbalance in the functionality which is offered by each component and fosters the modifiability of the architecture.

(d) pc – The propagation cost must be minimized due to indirect dependencies between components of an architecture affect the modifiability in a negative manner.

Figure 10 shows the four alternatives that can be obtained from the initial UI represented by $A22$. The architecture $A31$ incorporates a session component $S1$, replaces the previous twitter by $T2$ and connects both elements. $A32$ is similar to $A31$ but the session component $S2$ has an optional required interface for querying geo-localization information. The alternative $A33$ gathers a session component $S3$ and a geo-localization component $LC1$ in a container $C2$. In $A34$, the component $S4$ in $A34$ provides some geo-localization and weather information apart from the session functional interface.

In order to select the best model transformation alternative among $MT8$, $MT9$, $MT10$ and $MT11$, the bottom of Figure 10 depicts the values of the metrics calculated for $A31$, $A32$, $A33$ and $A34$. Note that components $M1$, $T2$, $L1$, $C2$, $S1$ and $S4$ are resizable, and components $LL2$, $S2$, $S3$,

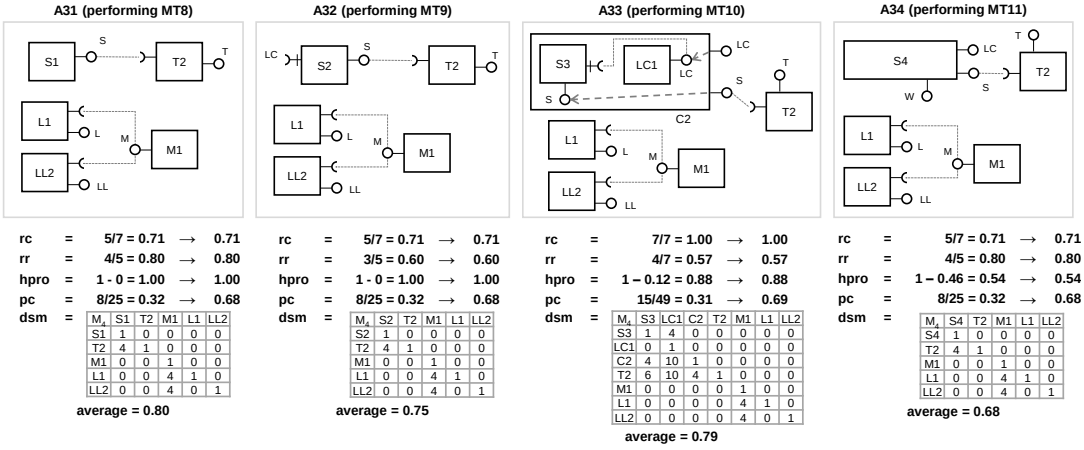

Figure 10. Transformation alternatives of scenario 3

with respect to the rest of metrics with the expression $npc = 1 - pc$ because in this scenario we try to maximize the customization of an architecture and the pc has a negative impact on his reconfiguration.

Finally, we select the best alternative by calculating the average of the four normalized metrics because in this scenario, the weights given for each metric and product factor are the same. As an example, the average for $A31$ is calculated from the math expression $(0.71+0.80+1.00+(1-0.32))/4$. Therefore, the resulting values for $A31$, $A32$, $A33$ and $A34$ are 0.80, 0.75, 0.79 and 0.68, respectively. As a consequence, we select $MT8$ as the best transformation that can be performed to adapt the UI and get the best customization value considering this subset of metrics in the transformation process.

6.4. Scenario 4: maximizing the cataloging

Continuing with the UI represented by $A31$, the next transformation process is intended to incorporate a new map into the workspace. Since the presence of the new map may generate confusion about what is the relationship between the layer list, the legend and the two maps, the components in the UI must be restructured accordingly. Figure 11 shows the three alternatives that can be reached from $A31$. The architecture $A41$ replaces the previous map $M1$ by $M2$ and uses $C3$ for containing it. In addition, $C3$ also contains $LL2$ and $L2$ components. The new map is resolved with an $M1$ component. The second alternative, $A42$, replaces the initial map $M1$ by $M3$, a map which includes the layer list and legend functionality. The new map is $M1$ type. The alternative $A43$ includes the same replacement of $A31$ but, in this case, the new map is $M2$ type and it is contained in a $C3$ component.

In this scenario, we want to maximize the correct use of the catalog of ENIA services (*i.e.*, cata-

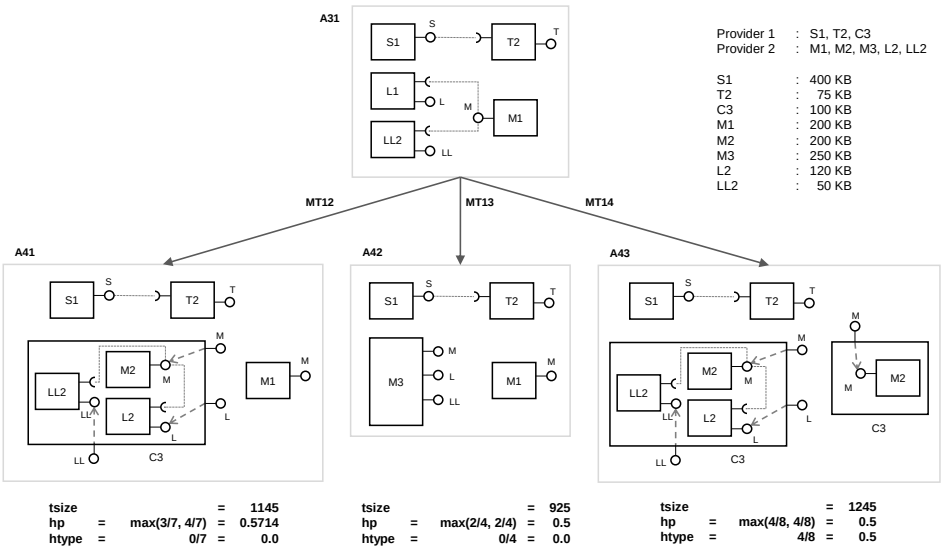

Figure 11. Transformation alternatives of scenario 4

to be managed as constraints to filter and select the best transformation. In this sense, we have also considered the size of the architecture to limit the maximum size of the alternative architectures:

(a) $tsize$ – The total size of components must be minimized and architectures with a value over 5MB will be rejected. Thus, we try to improve the performance efficiency of the browser by reducing the payload of the web components that must be initialized in the UI.

(b) hp – The homogenization among components' providers must be maximized because UIs with similar representation are preferred over components with heterogeneous representations. The use of the same provider does not guarantee the pursued homogenization, but the possibilities are greater if the entity providing the components is the same.

(c) $htype$ – The homogenization among components' types must be maximized because it is important to offer the maximum degree of consistency in the structure and representation of the UI's components. Therefore, components of the same type offer their functionality in the same manner.

Regarding the alternatives of Figure 11, each architecture accomplishes the best value for a different metric. In the case of $tsize$, the value of 925 MB from $A42$ is the best alternative. Focusing on hp, the best alternative is the architecture $A41$ because it gathers four components ($M1$, $M2$, $L2$ and $LL2$) from the same provider among the total of seven. With respect to $htype$, the best alternative is $A43$, because it contains four components ($M2$ * 2 and $C3$ * 2) having elements of the same type in the architecture. Therefore model transformation $MT12$ is chosen in the case of prioritize hp, whereas $MT13$ and $MT14$ are selected if $tsize$ and $htype$ are prioritized, respectively. Figure 12 shows the generated UI in the first case.

7. Discussions

This section discusses the benefits and drawbacks of applying the Quality-Aware transformation approach i

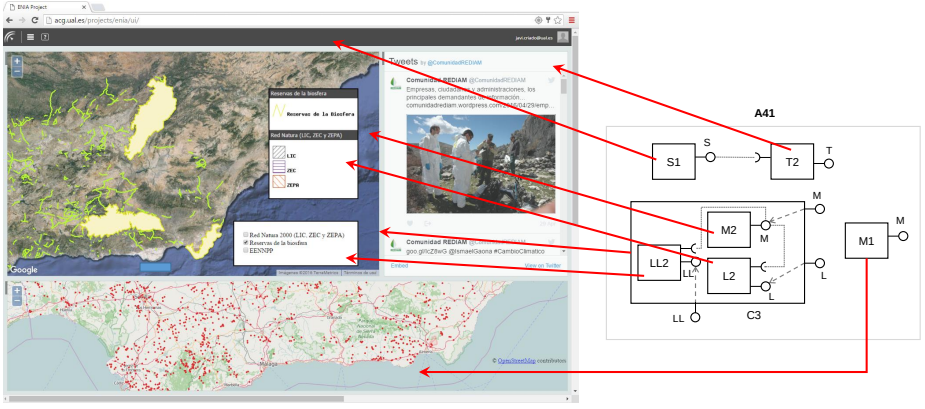

Figure 12. Resulting UI of scenario 4

7.1. What were the benefits and drawbacks of using the approach in ENIA?

Considering QAs at run-time improved the value of four quality assessments with regard to the generated architectures by model transformations in the ENIA case. The main advantage of using metrics related to QAs and constraints in ENIA is the incorporation of quality information in the process of selecting the best transformation operation that can be applied in UI adaptation. This allows us to use additional information (to functional interfaces) for solving the transformation process. In this sense, if these metrics are not applied, the transformation can generate architectures which may result in some drawbacks for the present use or future modifications.

For example, looking at the third scenario (see Figure 10), it is possible to obtain $A32$ as a solution instead of $A31$. In this case, we are 'loosing' the capability of having a session component which can be resized. On the contrary, using our approach we are able to offer 'resizability' of the components through the maximization of the rr metric. If we do not give the maximum priority to rr but we take it into account in the adaptation, at least the transformation at run-time will be enriched to improve the flexibility of generated UIs.

With regard to the future modifications, let us suppose that in the scenario 4 none of the metrics are applied and consequently, the generated transformation is equivalent to $MT13$ and the resulting architecture is $A42$. In this case, if the next adaptation step is aimed to remove the capability of selecting the layers to be displayed on the map (*i.e.*, LL provided interface), we faced two options: (1) the component $M3$ must be modified for hiding this interface and disabling its functionality, or (2) the component $M3$ must be replaced by other map which does not include this functionality, such as $M1$ or $M2$. In both options, we have to perform additional operations compared to those required in the case of starting the adaptation from the architectures $A41$ or $A43$, scenario in which we only should remove the component $LL2$.

Apart from these advantages, nothing is free in software engineering, and the performance of the QA-aware model transformation approach is an important trade-off that must be noted. Performance is related to the computation time necessary to (a) build each transformation alternative, (b) execute them obtaining the resulting architecture, and (c) measure each architecture to decide which transformation alternative is the best in terms of the quality information. The cost of these three execution times must be incorporated to the evaluation of the adaptation process described in [2] and, consequently, may not be possible to evaluate a large number of alternatives at run-time, having to limit the number of architectures evaluated to satisfy performance requirements. In addition, the size of the alternatives is limited to architectures whose size is smaller than twenty components [2]. Nevertheless the granularity of the components and the application domain imply that this is a valid limit size for our proposal.

7.2. Why to follow the Goal Question Metric Approach for steps 1 and 2?

One important difference between our previous work [4] and this paper is the way to conduct the analysis phase of our approach (*i.e.*, step 1).

On the one hand, our previous work studied the highest priorities of ENIA in an unsystematic manner with informal meetings with two software architects of ENIA. The important QAs and constraints were selected considering the perspective of two stakeholders, which stated that: "ENIA UIs must be reconfigurable and provide a friendly interaction by accomplishing the following objectives".

Therefore, the view of the quality in ENIA was partial, only identifying a subset of its important QAs. Among these QAs, we found: the modifiability attribute of the product quality model and the flexibility attribute of the quality in use model of the ISO/IEC 25010 standard, the total size of the UI, the homogenization of the components' providers, and the homogenization of the components' types.

On the other hand, in the current work we have conducted a software quality workshop designed at Fraunhofer IESE considering GQM+Strategies^{TM} [40], QUAMOCO [37], and GQM [33]. In the workshop a total of five stakeholders participated. Among the benefits, we could create an specific quality model of ENIA, and operationalize it with metrics.

8. Conclusions and Future Work

Software architects must take into account the functional requirements as the main issue to build software. Nevertheless, it is well accepted in the software architecture community that QAs are the most important drivers of architecture design [20]. Therefore, QAs should guide the selection of alternative software architectures from a model transformation process, considering the synergies and conflicts among them [42].

This work has analyzed how considering QAs at run-time can improve model transformation processes. Results in the ENIA case (a GIS interacted by mashup UIs), show that using a quality-aware architectural transformation at run-time can improve architectural-significant QAs. Four scenarios demonstrate how this approach can be applied to maximize the scope, the attractiveness, the customization or the cataloging, among other possible examples. The main contribution of this paper is a quality-aware transformation approach which consists of three steps: identifying relevant QAs and constraints, measuring them at run-time, and selecting the best alternative model transformation. To be able to perform this approach, first, a quality model has bee built from the application of the ISO/IEC 25010 standard and the QUAMOCO approach by applying the GQM strategy.

As future work, we will intend to perform more experimentations and reports in other adaptive domains besides mashup UIs. Moreover, we will study the possibility of handling the QAs during the generation of the alternative architectures to reduce the number of variants. In addition, a formal validation process in terms of execution times and model checking of the generated architectures could improve the proposed approach. The development of a tool supporting the execution of the QA-based transformation will support the automation of the proposed approach by selecting a subset of metrics that will be calculated depending on the pursued goal and the corresponding priority of metrics. Another open research line related to validation and evaluation issues is to carry out a controlled experiment with end users of ENIA and students of the software engineering course at University of Almeria and TU Kaiserslautern to study the impact of the proposed metrics (i.e., if they provide meaningful and correct assessments) and approach into the selection of candidate architectures or variants (i.e., if the approach is reliable). Such experiment would have two groups: one selecting the transformations based on relevant QAs, and another one selecting the transformations based on relevant QAs and the metrics that we propose for those.

Acknowledgments. This work was funded by the Spanish MINECO under TIN2013-41576-R and TIN2017-83964-R projects, and the ERCIM Fellowship Programme.

References

[1] Daniel F, Matera M. Mashups: Concepts, Models and Architectures. Springer-Verlag Berlin Heidelberg, 2014.

[2] Criado J, Rodríguez-Gracia D, Iribarne L, Padilla N. Toward the adaptation of component-based architectures by model transformation: behind smart user interfaces. *Software: Practice and Experience*, 2015. **45**(12):1677–1718. doi:10.1002/spe.2306.

[3] ACG. ENIA Project – Development of an intelligent web agent of environmental information. `http://acg.ual.es/projects/enia/`. Accessed: 2017-02-15.

[4] Criado J, Martínez-Fernández S, Ameller D, Iribarne L, Padilla N. Exploring Quality-Aware Architectural Transformations at Run-Time: The ENIA Case. In: Bellatreche L, Pastor Ó, Almendros Jiménez JM, Aït-Ameur Y (eds.), 6th International Conference on Model and Data Engineering, MEDI 2016, Almería, Spain, September 21-23, 2016, LNCS 9893, pp. 288–302. Springer International Publishing, 2016. doi:10.1007/978-3-319-45547-1_23.

[5] Salehie M, Tahvildari L. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.*, 2009. **4**(2):14:1–14:42. doi:10.1145/1516533.1516538.

[6] Carney D, Leng F. What do you mean by COTS? Finally, a useful answer. *IEEE Software*, 2000. **17**(2):83–86. doi:10.1109/52.841700.

[7] Bencomo N, Blair G. Using Architecture Models to Support the Generation and Operation of Component-Based Adaptive Systems. In: Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds.), Software Engineering for Self-Adaptive Systems, LNCS 5525, pp. 183–200. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-02161-9_10.

[8] Ameller D, Franch X, Cabot J. Dealing with Non-Functional Requirements in Model-Driven Development. In: RE 2010, 18th IEEE International Requirements Engineering Conference, Sydney, New South Wales, Australia, September 27 - October 1, 2010. IEEE Computer Society, 2010 pp. 189–198. doi:10.1109/RE.2010.32.

[9] Insfran E, Gonzalez-Huerta J, Abrahão S. Design Guidelines for the Development of Quality-Driven Model Transformations. In: Petriu DC, Rouquette N, Haugen Ø (eds.), Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part II, LNCS 6395, pp. 288–302. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-16129-2_21.

[10] Loniewsli G, Borde E, Blouin D, Insfran E. An Automated Approach for Architectural Model Transformations. In: José Escalona M, Aragón G, Linger H, Lang M, Barry C, Schneider C (eds.), Information System Development: Improving Enterprise Communication, pp. 295–306. Springer International Publishing, 2014. doi:10.1007/978-3-319-07215-9_24.

[11] Ameller D, Franch X, Gómez C, Araujo J, Svensson RB, Biffl S, Cabot J, Cortellessa V, Daneva M, Fernández DM, Moreira A, Muccini H, Vallecillo A, Wimmer M, Amaral V, Brunelière H, Burgueño L, Goulão M, Schätz B, Teufl S. Handling non-functional requirements in Model-Driven Development: An ongoing industrial survey. In: 23th IEEE International Requirements Engineering Conference (RE), pp. 208–213. IEEE Computer Society, 2015. doi:10.1109/RE.2015.7320424.

[12] Criado J, Martínez S, Iribarne L, Cabot J. Enabling the Reuse of Stored Model Transformations Through Annotations. In: Kolovos D, Wimmer M (eds.), Theory and Practice of Model Transformations: 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings, LNCS 9152, pp. 43–58. Springer International Publishing, 2015. doi:10.1007/978-3-319-21155-8_4.

[13] Solberg A, Oldevik J, Aagedal JØ. A Framework for QoS-Aware Model Transformation, Using a Pattern-Based Approach. In: Meersman R, Tari Z (eds.), On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004, Agia Napa, Cyprus, October 25-29, 2004, Proceedings, Part II, LNCS 3291, pp. 1190–1207. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30469-2_25.

[14] Weyns D, Ahmad T. Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review. In: Drira K (ed.), Software Architecture: 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings, LNCS 7957, pp. 249–265. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-39031-9_22.

[15] Ozkaya I, Nord R, Koziolek H, Avgeriou P. Second Int. Workshop on Software. Architecture and Metrics. In: ICSE'2015, pp. 999–1000. IEEE Press, 2015.

[16] Carriere J, Kazman R, Ozkaya I. A cost-benefit framework for making architectural decisions in a business context. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, volume 2, pp. 149–157. 2010. doi:10.1145/1810295.1810317.

[17] Martínez-Fernández S, Ayala CP, Franch X, Marques HM. REARM: A Reuse-Based Economic Model for Software Reference Architectures. In: Favaro J, Morisio M (eds.), Safe and Secure Software Reuse: 13th International Conference on Software Reuse, ICSR 2013, Pisa, June 18-20. Proceedings, LNCS 7925, pp. 97–112. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38977-1_7.

[18] Stevanetic S, Javed MA, Zdun U. Empirical Evaluation of the Understandability of Architectural Component Diagrams. In: Proceedings of the WICSA 2014 Companion Volume, WICSA'14 Companion, pp. 4:1–4:8. ACM, 2014. doi:10.1145/2578128.2578230.

[19] Zimmermann O. Metrics for Architectural Synthesis and Evaluation – Requirements and Compilation by Viewpoint. An Industrial Experience Report. In: 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, pp. 8–14. 2015. doi:10.1109/SAM.2015.9.

[20] Ameller D, Ayala C, Cabot J, Franch X. Non-functional Requirements in Architectural Decision Making. *IEEE Software*, 2013. **30**(2):61–67. doi:10.1109/MS.2012.176.

[21] Ameller D, Franch X. Assisting software architects in architectural decision-making using Quark. *CLEI Electron. J.*, 2014. **17**(3). URL `http://www.clei.org/cleiej/papers/v17i3p1.pdf`.

[22] Guzmán L, Oriol M, Rodríguez P, Franch X, Jedlitschka A, Oivo M. How Can Quality Awareness Support Rapid Software Development? - A Research Preview. In: Requirements Engineering: Foundation for Software Quality - 23rd International Working Conference, REFSQ 2017, Essen, Germany, February 27 - March 2, 2017, Proceedings, LNCS 10153, pp. 167–173. Springer International Publishing, 2017. doi:10.1007/978-3-319-54045-0_12.

[23] Abdellatief M, Sultan ABM, Ghani AAA, Jabar MA. A mapping study to investigate component-based software system metrics. *Journal of Systems and Software*, 2013. **86**(3):587 – 603. doi:10.1016/j.jss.2012.10.001.

[24] Goulao M, Abreu FB. Composition assessment metrics for CBSE. In: 31st EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 96–103. IEEE Computer Society, 2005. doi:10.1109/EUROMICRO.2005.19.

[25] Rotaru OP, Dobre M. Reusability metrics for software components. In: The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005., pp. 24–31. IEEE Computer Society, 2005. doi:10.1109/AICCSA.2005.1387023.

[26] Vieira MER, Dias MS, Richardson DJ. Describing Dependencies in Component Access Points. In: Procs of the 23rd Intern. Conf. on Software Engineering, ICSE'01, pp. 115–118. 2001.

[27] Bertoa MF, Vallecillo A. Quality Attributes for COTS Components. In: 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002), pp. 1–11. 2002.

[28] Bertoa MF, Troya JM, Vallecillo A. Measuring the usability of software components. *Journal of Systems and Software*, 2006. **79**(3):427 – 439. doi:10.1016/j.jss.2005.06.026.

[29] Voas J, Payne J. Dependability certification of software components. *Journal of Systems and Software*, 2000. **52**(2–3):165 – 172. doi:10.1016/S0164-1212(99)00143-0.

[30] Washizaki H, Yamamoto H, Fukazawa Y. A metrics suite for measuring reusability of software components. In: Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717), pp. 211–223. IEEE Computer Society, 2003. doi: 10.1109/METRIC.2003.1232469.

[31] Boxall MAS, Araban S. Interface metrics for reusability analysis of components. In: 2004 Australian Software Engineering Conference. Proceedings., pp. 40–51. IEEE Computer Society, 2004. doi:10.1109/ASWEC.2004.1290456.

[32] Narasimhan VL, Hendradjaya B. Some theoretical considerations for a suite of metrics for the integration of software components. *Information Sciences*, 2007. **177**(3):844 – 864. doi:10.1016/j.ins.2006.07.010.

[33] Basili VR, Caldiera G, Rombach HD. The Goal Question Metrics Approach. Encyclopedia of Software Engineering. Wiley, 1994.

[34] ISO/IEC. ISO/IEC 25010. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models. 2011.

[35] Kazman R, Klein M, Clements P. ATAM : Method for Architecture Evaluation. *Cmusei*, 2000. **4**(August):83. doi:(CMU/SEI-2000-TR-004,ADA382629). URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6014{\&}rep=rep1{\&}type=pdf`.

[36] Bellomo S, Gorton I, Kazman R. Toward Agile Architecture: Insights from 15 Years of ATAM Data. *IEEE Software*, 2015. **32**(5):38–45. doi:10.1109/MS.2015.35. URL `http://ieeexplore.ieee.org/document/7024074/`.

[37] Wagner S, Goeb A, Heinemann L, Kläs M, Lampasona C, Lochmann K, Mayr A, Plösch R, Seidl A, Streit J, Trendowicz A. Operationalised product quality models and assessment: The Quamoco approach. *Information & Software Technology*, 2015. **62**:101–123. doi:10.1016/j.infsof.2015.02.009.

[38] Ramírez A, Romero JR, Ventura S. An approach for the evolutionary discovery of software architectures. *Information Sciences*, 2015. **305**:234 – 255. doi:10.1016/j.ins.2015.01.017.

[39] Criado J, Iribarne L, Padilla N, Ayala R. Semantic Matching of Components at Run-Time in Distributed Environments. In: Ciuciu I, Panetto H, Debruyne C, Aubry A, Bollen P, Valencia-García R, Mishra A, Fensel A, Ferri F (eds.), On the Move to Meaningful Internet Systems: OTM 2015 Workshops: Confederated International Workshops, Rhodes, Greece, October 26-30, 2015. Proceedings, LNCS 9416, pp. 431–441. Springer International Publishing, 2015. doi:10.1007/978-3-319-26138-6_46.

[40] Basili VR, Trendowicz A, Kowalczyk M, Heidrich J, Seaman CB, Münch J, Rombach HD. Aligning Organizations Through Measurement - The GQM+Strategies Approach. The Fraunhofer IESE Series on Software and Systems Engineering. Springer International Publishing, 2014. doi:10.1007/978-3-319-05047-8.

[41] Crnkovic I, Sentilles S, Vulgarakis A, Chaudron MRV. A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*, 2011. **37**(5):593–615. doi:10.1109/TSE.2010.83.

[42] Boehm B. Architecture-Based Quality Attribute Synergies and Conflicts. In: 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, pp. 29–34. 2015. doi:10.1109/SAM.2015.18.