Toward the adaptation of component-based architectures by model transformation: behind smart user interfaces

Javier Criado*,†, Diego Rodríguez-Gracia, Luis Iribarne and Nicolás Padilla

Applied Computing Group, Department of Informatics, University of Almería, Spain

SUMMARY

Graphical user interfaces are not always developed for remaining static. There are GUIs with the need of implementing some variability mechanisms. Component-based GUIs are an ideal target for incorporating this kind of operations, because they can adapt their functionality at run-time when their structure is updated by adding or removing components or by modifying the relationships between them. Mashup user interfaces are a good example of this type of GUI, and they allow to combine services through the assembly of graphical components. We intend to adapt component-based user interfaces for obtaining smart user interfaces. With this goal, our proposal attempts to adapt abstract component-based architectures by using model transformation. Our aim is to generate at run-time a dynamic model transformation, because the rules describing their behavior are not pre-set but are selected from a repository depending on the context. The proposal describes an adaptation schema based on model transformation providing a solution to this dynamic transformation. Context information is processed to select at run-time a rule subset from a repository. Selected rules are used to generate, through a higher-order transformation, the dynamic model transformation. This approach has been tested through a case study which applies different repositories to the same architecture and context. Moreover, a web tool has been developed for validation and demonstration of its applicability. The novelty of our proposal arises from the adaptation schema that creates a non pre-set transformation, which enables the dynamic adaptation of component-based architectures.

KEY WORDS: component-based architectures; run-time adaptation; model transformations; higher-order transformations; mashup user interfaces; smart user interfaces

1. INTRODUCTION

Software systems and applications dealing with user interaction need UI to achieve the communication. The most widely used UIs are the GUI, regardless of the platform utilized for accessing. As other existing software artifacts, many GUIs are not intended to remain static from the moment they are designed. On the contrary, these GUIs implement in their behavior some variability mechanisms (e.g., execution alternatives), configuration operations (e.g., settings), or an external system that is in charge of modifying their internal structure (e.g., varying the code of non-compiled GUIs).

The case of the web platform is a frequently changing domain and thus web UIs are subject to constant variations associated with the resources or services which are accessed. For this reason, this kind of UIs benefit from a dynamic behavior. In this sense, *mashup* UIs are a good example of combining services with the aim of composing a single interface to interact with them [1]. Therefore, the customization of these UIs through the configuration of the services that the user wants to visualize is an issue that has been addressed in the literature [2].

Mashup UIs are built from graphical components of medium and high granularity (not simple text fields or buttons) that encapsulate some functionality. This kind of GUIs can be used in different

*Correspondence to: Javier Criado, Applied Computing Group, Department of Informatics, University of Almería, Spain.
†E-mail: javi.criado@ual.es

domains. For example, in *geographic information systems*, these interfaces allow us to exploit information from different maps, geospatial services, or other geographic data resources [3]. In *enterprise resource planning* applications, each user profile can be related with a specific UI structure and with the UI pieces that are available to perform either different or common tasks [4]. Other examples are *dashboard* UIs, which bring together different graphical components from different sources [1]. These components have many different purposes (e.g., RSS, social networks, weather information, activity of a web site, etc.), and the UIs can be used by individual users or a group of them, as implemented in Netvibes, MyYahoo, or Ducksboard.

Nevertheless, the current proposals of these GUIs built from pieces could be improved in some ways. First, the different components of the interface are isolated and there are no dependencies between them. Therefore, the interaction in one component has no impact on other components. Otherwise, the re-configuration of the GUIs not only depends on the manual setting by the users (customization perspective) but can also be interesting performing an automatic adaptation from the system decisions (proactive perspective) [5]. An example of the second possibility may occur if two users are working together to solve a common task. Then, at a certain moment in the task, it becomes necessary to communicate with each other. The system detects this action, and their UIs can self-adapt (without being requested by the actor), incorporating a new *chat* component (i.e., to their inner architectures) that allows them to interact [6].

1.1. Solution overview

Previous aspects motivate us to study the run-time adaptation of component-based GUIs. Actually, this is an open research topic included in two projects of the Spanish Ministry and the Andalusian Government. Our main interest in this domain is due to the trend toward the GUI *Social Semantic Web* or *Web 3.0* [7, 8], a mixture of *Social Web* [9] and *Semantic Web* [10] topics. Under this trend, UIs make use of technologies to favor information exchange within the Web community, such as the social networks; they also incorporate certain intelligence capabilities that allow users to share common purposes to cooperate and integrate data by means of standards, which allow more open applications. Under this context (SSW-based systems), we know that it could be useful in this kind of systems because it is possible to define 'component-based UIs' that can self-adapt their structure depending on the run-time circumstances. Thus, these interfaces will offer new functionalities or will hide others in order to fit the requirements (Figure 1). As a final goal, we intend to establish the infrastructure for obtaining 'smart GUIs' or what we call *SmartGUI* (SUI), intelligent UIs which learn from the interaction of/with the user or group of users, re-configuring their components (i.e., their inner architecture).

Consequently, our approach is not valid for all sorts of GUI. Our proposal requires that the UI has to be defined as an architectural model, in which each component of the architecture represents an individual UI component. We follow a *bottom-up* perspective for the (re)building (at run-time) of the structure of the UI from those GUI components fulfilling the requirements of the architecture, available in one or more third party repositories. In our methodology, UI components are called COTSgets [11], a combination of the terms commercial off-the-shelf (*COTS*) [12] and *widget* (which is how this type of UI component is usually known). Unlike other 'widget-based UI' proposals, our UI components may have interdependencies which may affect the system behavior. The adaptation process therefore changes the structure of the UI based on the transformation of its associated architectural model. The adaptation is determined by changes in the context at run-time, for example, the user interaction, changes in the system requirements, and time events.

Therefore, it is necessary to provide these architectures with certain adaptation mechanisms to evolve its behavior automatically, making use of concepts related with *self-adaptive systems* (SAS) [13]. *Component-based software engineering* can be of assistance in developing a self-adaptive system. It provides certain advantages in SAS [14], for instance, the modularization and reuse of software components, or the use of component models is well-suited for dynamic incorporation of new services [15]. New proposals arise through the track of self-adaptive software and the goal of overcoming the limitations of static system models, attempting to adapt the software system

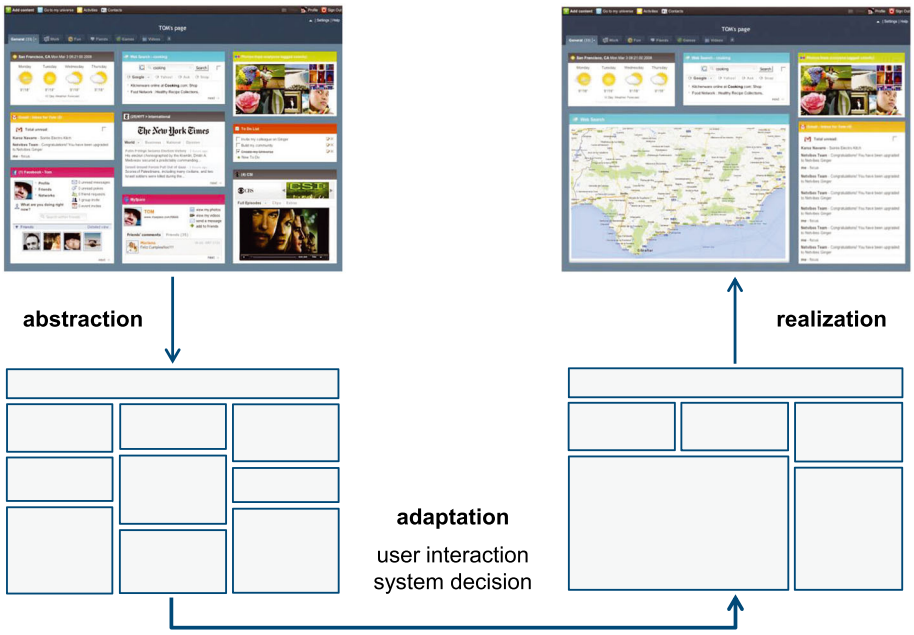


Figure 1. GUI adaptation.

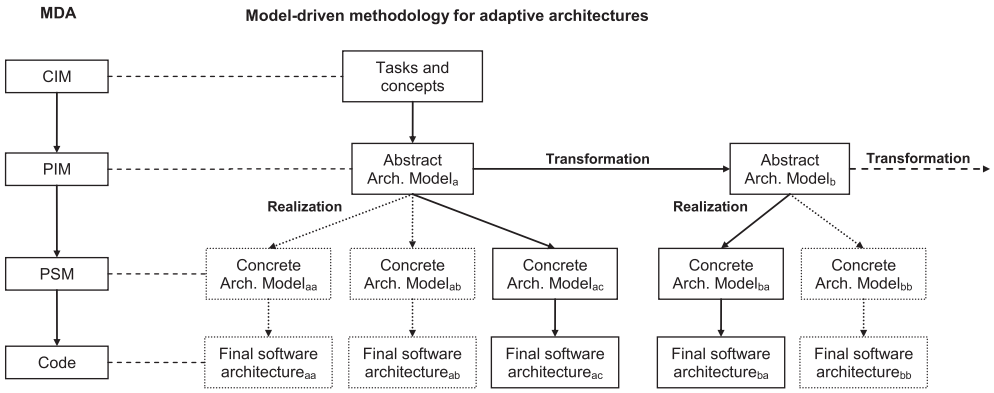


Figure 2. Model-driven methodology.

automatically by manipulating the models of the system [16]. This new vision makes use of *model-driven engineering* (MDE) concepts and adds functionalities for adapting the models at run-time. In the particular domain of *component-based software systems*, the use of MDE techniques can facilitate the design and development of architectures, for example, for defining their structure, the behavior of their components and relationships, their interaction, or their functional and non-functional properties [17]. Furthermore, the manipulation of architectural models at run-time makes it possible to generate different software systems based on the same abstract definition, adapting, for example to user interaction, component status, or execution platform [18].

In the proposed methodology, the life cycle for developing component-based architectures is structured on four levels of abstraction, inspired by the Cameleon reference framework [19] which is also based on the Object Management Group (OMG) *model-driven architecture* (MDA) specification [20] (see Figure 2): (i) *tasks and concepts*, which corresponds to the *computation independent model* level of MDA and represents the tasks that must be performed to meet the system requirements; (ii) *abstract architectural model*, which corresponds to the *platform independent model* (PIM) level and represents the architecture in terms of what type of components it contains and their rela-

tionships; (iii) *concrete architectural model*, which corresponds to the *platform specific model* (PSM) level and describes what concrete components comply with architectural abstract definition; and (d) *the final software architecture*, which represents the source code that will be executed or interpreted.

1.2. Technological and conceptual context

The adaptation of the architectures is done based on processes executed on the *abstract* and *concrete* architectural levels [21, 22]. On the abstract level, *model-to-model* (M2M) transformation processes [23] are executed to change and adapt the abstract architectural models to the changes in context. However, the concrete architectural models are realized by a trading process [12], calculating the configurations of concrete components that best meet the abstract definitions. This provides the possibility of generating different software architectures based on the same abstract definition, for example, so it can be executed on different platforms.

This methodology isolates the abstract GUI model (PIM view) from the device (PSM view). Therefore, this schema facilitates the adaptability of GUIs not only regarding behavior and functionality but also for different devices (PC, SmartPhone, tablet, SmartTV, etc.) by solving the concrete model based on the abstract one. Furthermore, we think that this proposal could be useful in a diversity of component-based systems, for example, *smart home applications* [24, 25], *smart TV* [26], *smart cars* [27], *smart buildings* (or *intelligent buildings*) [28], *smart cities* [29], *robotics* [30, 31], *communication network infrastructures* [32], and *UIs* [33]; in summary, any component-based software that must be adapted at run-time and whose components are interrelated.

For this reason, this paper is aimed at describing the adaptation process in a generalist way, that is, defining the architectures and the rest of involved elements regardless of the domain chosen as a case study for its application. Within the complete adaptation methodology, we want to remark that this paper focuses only on the adaptation performed on the abstract level (PIM perspective), but not on the trading process that obtains the concrete architectures (PSM), that is, the *transformation* step in Figure 2. Similarly, this paper does not discuss synchronization issues between abstract models and final architectures or how the changes in the models affect the executing architecture. However, we understand that our proposal is more than a *rule-based* approach for dynamic adaptation. Some of the adaptation operations are performed on an abstract level instead of running systems. But these operations make sense because then, at the concrete level, our trading process will choose different concrete components from the abstract definition of the architecture, making possible that this trading process (in the future) will be able to choose a concrete component or another depending on the platform (PSM perspective) and other aspects such as concrete component properties. The proposal presented in this article has been developed based on the concepts established in [22], and it significantly improves the adaptation process described in [34].

On the abstract level, M2M transformations are used to adapt the architectural models dynamically. Traditionally, the M2M transformation logic is static, described a priori by rules that are embedded in the code. This behavior impedes models from adapting to the requirements that were not taken into account in the system design stage. Our proposal is intended to make the transformations dynamically, so they also change, adapting to the new requirements and variations in context. Therefore, transformations are built at *run-time*, selecting from a rule repository those transformation rules that are suitable to the current situation. Inspired by the idea of co-evolution [35], in which the changes in the metamodels defining the system provoke the corresponding adaptation in the models they define, our proposal refactors [36] M2M transformations that adapt the architectural models at run-time. This process, which we call *co-transformation* of the M2M processes that adapts the architectures, has been implemented following a *higher-order transformations* (HOT) perspective [37]. It is a HOT-type transformation because it generates as output a new M2M transformation, which is responsible for adapting the architectural models.

The rest of the paper is organized as follows. Section 2 describes the goals and fundamentals and defines the main concepts of our proposal. Section 3 defines the adaptation process formalisms and the adaptation schema. Section 4 explains in-depth the adaptation methodology, and it also provides an example of the aforementioned process using a case study. Section 5 shows the validation and

evaluation processes performed on our proposal and discusses the benefits and shortcomings of the approach. Section 6 presents the related work, and finally, Section 7 gives some conclusions and future work.

2. ESSENTIALS OF ADAPTATION PROCESS

In this section, we attempt to explain the principles on which our proposal is based. The first subsection enumerates the intended *goals*. In the following subsections, the key pieces in our approach are presented, that is, how we define our *architectures*, how the *components* are described, what role the *models* play in defining the architectures at design-time and adapting them at run-time, how we have defined and how we execute our *adaptation*, and how we achieve this adaptation by making use of *model transformations*.

2.1. Contributions of the proposal

As we introduced, we thought that our approach could be generalized to any type of software architecture with component interdependencies not just to component-based UIs. Therefore, the general goals pursued and the contributions of this approach are the following:

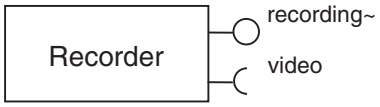
- Define our software architectures by means of a component-based representation.
- Develop an adaptation process for architectures which is not static, that is, one that is flexible and has an adaptive logic that can dynamically change.
- Define the logic of rule-based adaptation that achieves a dynamic adaptive process by varying the rules that can be applied depending on the system context.
- Establish a representation for the adaptation rules so that these rules can be stored in a repository and subsequently, managed and selected for their application.
- Define a mechanism for selecting the adaptation rules at run-time depending on the information from the context and the system requirements.
- Define an adaptation engine to apply the rules selected, thereby changing the structure of the architecture.

The aforementioned goals cover some basics of traditional software architectures processes [38], such as design, representation, and a purpose for future realization. Nevertheless, other features more related to self-adaptive software (e.g., definition of changes, reconfiguration options, or transformation alternatives) [39] are also addressed. In this sense, with these overall goals, we are able to adapt component-based software architectures at run-time. Furthermore, the process that adapts these architectures is not pre-set but dynamically 'constructed' based on the rules selected according to the state of the context, the system requirements, and the adaptation purpose. This makes it possible for the architecture to be adapted by updating or changing the rule repository. As stated in [39], the ability to select different types of transformations provides expressiveness to adaptive systems. The mentioned goals will provide our approach with a constrained selection process from a predefined set of rules, which are stored in the repository. Nevertheless, our system is ready to perform an unconstrained selection process because we will add and remove rules of/from the transformation rule repository.

2.2. Components

As mentioned in the introduction, the component-based software architectures in our system are defined on two levels: *abstract* and *concrete*. However, the approach described in this article is focused only on adaptation on the abstract level, so whenever we refer to architectures or components in the following text, they are on the abstract level. The following definition defines what an *abstract component* is.

Definition 1 (Abstract component)
An abstract component is the type of component that forms part of the abstract architecture. That is, it contains the definition of the component based on its functional and non-functional properties.



functional_properties
provided_interfaces: recording
required_interfaces: video

non_functional_properties
output_format: FLV
resolution_width: 800
resolution_ height: 600
resolution_unit: px

Figure 3. An example of abstract component.

Functional properties refer to the component ports which require or provide a series of interfaces. Non-functional properties describe component characteristics that are not related directly to the services they offer or that are required for their functioning but provide solutions for component search, evaluation, and selection activities.

For example, as shown in Figure 3, the abstract component named as `Recorder` contains two ports in its *functional* part. One port corresponds to the provided interface, `recording`, whereas the other implements the required interface, `video`. The provided interface identifies the services offered by the component, whereas the required interface defines the service this abstract component requires to function. The component has four properties in the *non-functional* part: one for the output video format (`output_format`), which must be `FLV`, one for the video capture resolution (`resolution_unit`), which is `px`, and two properties that describe the width (`resolution_width`) and height (`resolution_height`) of the recording resolution.

2.3. Architectures

This subsection illustrates the difference between the abstract and concrete levels. The following definition describes what an *abstract architecture* is.

Definition 2 (Abstract architecture)
An abstract architecture defines the set of abstract components which are in (or should be in) the architecture to function properly. An abstract architecture also contains information about how the component ports are interconnected.

For instance, at the top of Figure 4, there is an example of abstract architecture describing a UI comprised of four components: a `Chat` component type, an `Audio` component type, a `Video` component type, and a `Recorder` component type, which requires the interface provided by video (in order to capture images) to work. These four components are contained in the `GUI` component type. At the bottom of the same figure, there is a concrete architecture corresponding to the aforementioned abstract definition. A concrete architecture describes the concrete components that best meet the abstract definition and have been selected from among all the candidates. For further details on the concrete level of architectures, see [22, 40]. In this case, the `JSP_Container` concrete component is selected to resolve the `GUI` component type. Otherwise, four concrete components are selected to realize the rest of the abstract definitions: `JABBERChat`, `DirectShowAudio`, `VISCOMVideo`, and `VISCOMRecorder`.

2.4. Models

Our proposal is based on MDE, so models are an essential element in our research work. In the first place, our architectures are defined by means of models. Although there is a large number of *architectural description languages*, for our system, we preferred to construct our own *domain-specific*

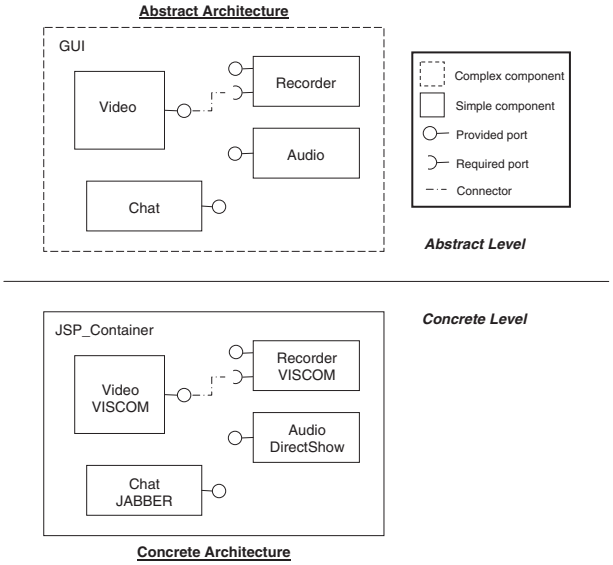


Figure 4. An example of abstract and concrete levels of our architectures.

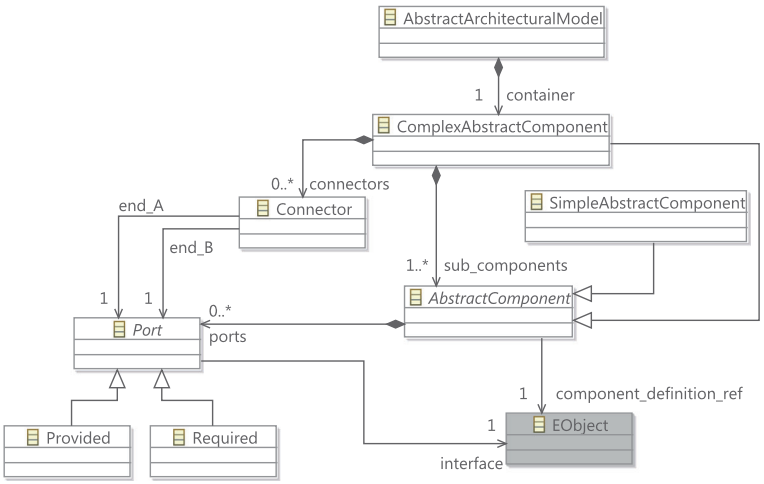


Figure 5. Architectural metamodel.

language (DSL). There are numerous advantages of using metamodeling in design and implementation of component-based systems [41]. The main reason is that our DSL defines exactly the elements we need to manage our system and simplifies manipulation and operations with our architectural models. For example, in our architectures, each component port is related only with one interface. This decision was made for two reasons: (i) we need component ports to manage the addressing of its interfaces; and (ii) because the new architectures are created at run-time, the easiest way to generate new ports is performing a one-to-one correspondence with the interfaces. Moreover, each component of the architecture has a reference to a component model describing this particular element. In our case, our components are described by using functional and non-functional properties, inspired by the commercial off-the-shelf component model [12]. These features encourage us to create our own architectural and component metamodels, which will be described as follows.

Construction of the DSLs in our system was done following the *meta-object facility* standard [42] using the *eclipse modeling framework* (EMF) [43]. This way, the constructed metamodels provide the expressiveness necessary to define the architectural elements. Figure 5 shows the metamodel

for defining abstract architectures. An architectural model is comprised of a `container` as the root element. This container is a `ComplexAbstractComponent`, which in turn is composed of elements that can be simple or complex. Components have ports that implement `Provided` or `Required` interfaces. The related interface of each port is defined by the `interface` reference. These ports are connected by `Connector` elements contained in the complex component which is the parent of the components it connects. Connections represent the dependencies between components; therefore, component A depends on another component B when there is a connection between a required port of the first component and a provided port of the second one. Finally, the architectural components have a reference to the model that defines them in `component_definition_ref`.

The reference named `component_definition_ref`, included in the architectural metamodel, relates each component in the architectural model with the definition of this component in a repository of components. Similarly, the `interface` reference links each `Port` from the architectural model to the corresponding component `Interface`. Both references have been depicted in Figure 5 as an `EObject` element for representing the link with the respective element of the model describing the repository. This repository and its components were also constructed using metamodeling. Figure 6 shows the metamodel for defining the components of the system. Each component has a `Functional` part and optionally, a `NonFunctional` part. The first contains the functional system properties, which are distinguished between provided and required. These functional properties correspond to the provided and required ports mentioned earlier, which are used to represent the dependencies between the components of an architecture. The second contains non-functional properties of the component. Apart from the models used to represent the architectures and components, in our system, the models participate in the definition of other important parts.

- Description of the context information observed as input for the adaptation process (from now on called *observer model*).
- List of adaptation operations found to be necessary after context information processing (called adaptation *operation model*).
- Definition of the rules that are run to adapt the architectural models (stored in the *rule repository model* and the *selected rule model*).

In addition to the architectural and component models, these models are also an indispensable part of the system. All of the models together determine system functioning, as the different views used in *orthographic software modeling* [44] do. This division in system definition, also inspired by the concept of *Megamodels* [45], facilitates management of models and their relationships in

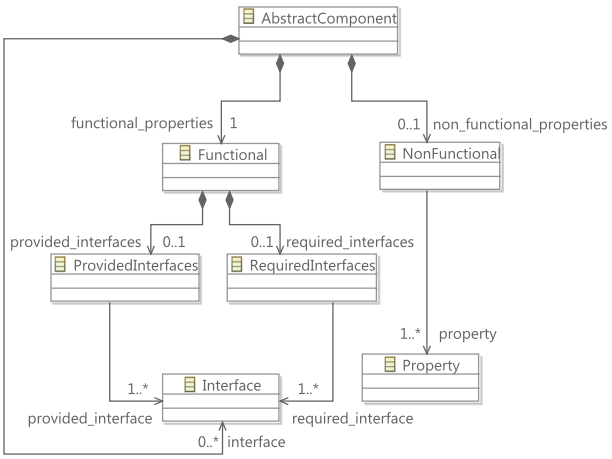


Figure 6. Component metamodel.

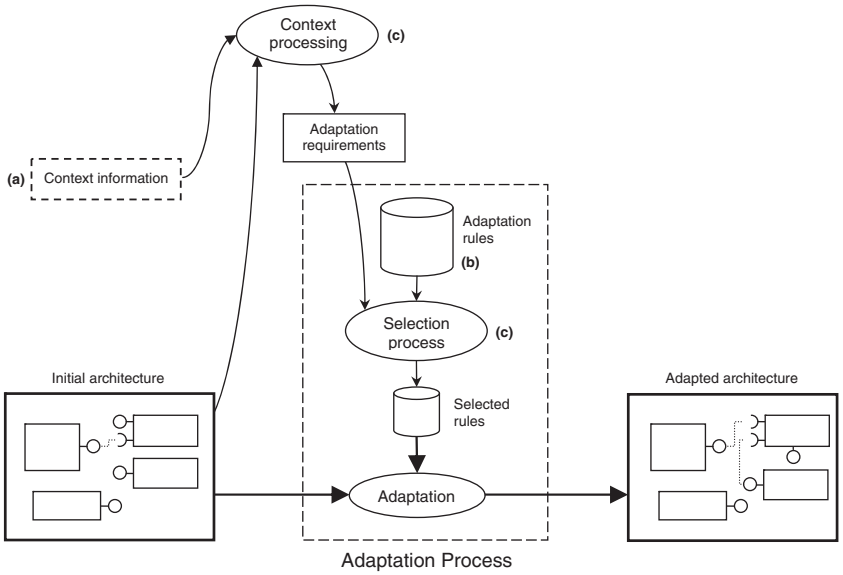


Figure 7. Adaptation rules.

adapting the system at run-time [46]. Because of space constraints, we are not able to show all the metamodels involved in our proposal. Nevertheless, metamodel definitions and some model examples are available in our website [47][‡].

The use of models in the definition of the parts of the system allows us to formally describe the syntax and semantics of the elements involved in our adaptation process. Moreover, the description of the models according to their metamodels using EMF, allows us to define constraints using *object constraint language* (OCL) [48]. This feature enables us to perform validation operations checking that the OCL constraints and the structural definitions specified in the metamodel are fulfilled.

2.5. Adaptation rules

Traditional proposals for M2M transformation define the transformation logic by means of rules that are hard-coded [23]. Nevertheless, there are some model transformation languages and some approaches with mechanisms for model transformation reuse, such as rule-based modularization [49] or module composition [50]. In contrast, the actions that adapt our architectures are collected in a repository to make adaptation more dynamic. The purpose of this rule repository, which is defined by a model as mentioned in Section 2.4, is to assemble the set of possible operations that can be executed. Each adaptation rule has an associated *action*, a *weight*, and a Boolean attribute that shows its *priority*, in addition to other attributes. This way, a *selection process* can be constructed based on these attributes and the adaptation necessary at any given instant by choosing the rules that best meet the purpose of adaptation. Figure 7 shows a schematic diagram of how this selection process works.

Representing our adaptation rules this way and defining the selection process provides a mechanism for *variability* that can obtain different adapted architectural models from the same starting architectural model. The resulting adapted model depends on (i) the context information that influences determination of system adaptation requirements; (ii) the repository of adaptation rules; and (iii) the logics defined in rule selection and context processing. The *first* approach to the proposal, in which we considered fixed and invariable rule repository, selection logic, and context processing, resulted in an adaptation process that generated different architectural models depending on the system context information. However, in the *second* and improved version, the rule repository could change, and given identical adaptation requirements, the system could generate different architec-

[‡]In this web page, we included all the material developed for this contribution: models, metamodels, source code of model transformations, and the extraction process, http://acg.ual.es/isoleres/adaptation.

tural models depending on the adaptation rule repository. This made it possible to modify, add, and erase repository rules to get different adaptive behaviors. In the *third* approach, the logic of rule selection and the logic determining adaptation requirements can vary, making adaptation more dynamic (and itself more adaptable) at run-time.

The purpose of this article is to describe this approach for architectural model adaptation up to the second version, that is, the selection logics and context processing are predefined. Therefore, the context information and the rule repositories are modified in order to show the variability achieved and possible future improvements in adaptation.

2.6. Model transformation

Our goal is to adapt software architectures that are described by models, therefore, there must be a mechanism that can execute operations adapting these models. This mechanism is well represented by *model transformation*, in particular, by M2M transformation. Model transformation is an essential part of MDE, because it makes it possible to manipulate models based on their definition and the restrictions imposed on them [51]. In an M2M transformation, model A is transformed into model B based on certain mapping relationships among its elements or other operations as described by the transformation behavior itself [52].

Our proposal adapts a starting architectural model (AM_A) into another adapted architectural model (AM_B), both defined by the same architectural metamodel AMM, based on these transformation processes. To do this, we defined an M2M transformation containing a series of model transformation rules. In a later adaptation step, due to certain changes in the context or system requirements, model AM_B will have to be transformed into a new architectural model AM_C. The behavior of this new model transformation may be the same as in the previous step, similar or completely different, so there are two possibilities: (i) define all the possibilities for adaptation model transformation existing a priori, for example, using state machines [21] or describing different transformation alternatives [53]; or (ii) transformation behavior is not pre-set and can be defined dynamically depending on the system requirements. This article concentrates on defining an adaptation schema that provides a solution for the second option. Figure 8 shows an *adaptive transformation* with behavior that is not always the same and varies to generate different architectural models.

Apart from this, our use of model transformation is not limited to adapting architectural models. As all the pieces that participate in the adaptation schema can also be found in the field MDE, all their elements are defined and represented by models, and all the operations that are executed on them are done by M2M transformations.

Admittedly, it is not indispensable to use model transformations techniques for performing transformation operations to be responsible for adapting component-based systems. However, we want to apply MDE techniques to these systems, as model transformations allow us to *formally build* output elements that conform to the output metamodel, generate *traceability* information about the performed operations, and include operations for *validating and checking the consistency* within the model transformation. Therefore, we can use these techniques to ensure the generation of architectural models that comply properly with their definition.

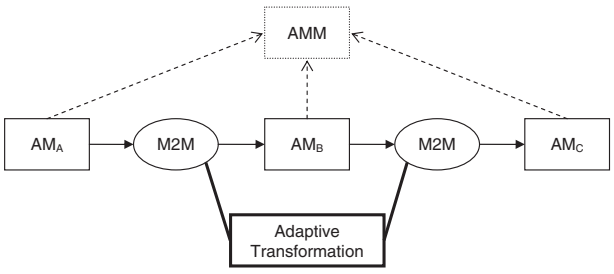


Figure 8. Adaptive model transformation.

3. ADAPTATION PROCESS

Our proposal attempts to achieve adaptation of architectural models at run-time. In the beginning, these models are defined at design-time and have to change to adapt to later changes in system context. The models generated at design-time are static artifacts so there must be a mechanism for transforming them. In our case, we follow an MDE methodology not to generate the final static code based on models but to represent our architectural models and adapt them at run-time.

This adaptation of the architectural models is done by M2M transformation. In fact, our adaptation process is, schematically, a sequence of transformations. As all the elements that participate in the transformation sequence have similar characteristics, we were able to develop a DSL to define the adaptation abstraction based on the established principles. In this section, we first describe an overview of the entire process, then we formalize the description of the elements that compose it. Finally, we define the relationships between these elements through the previous formalization, resulting in our adaptation schema. Furthermore, for the explanation of the performed process, our methodology is described in Section 4 through a running example.

3.1. Adaptation schema

With the aim of showing an overall view of the transformation sequence, Figure 9 shows each of the elements (models, metamodels, transformations, extractions, and relationships) involved in the adaptation process. To summarize, our proposal could be described as a technique made up of M2M transformations which start out from a starting architectural model, generating a new one at run-time that seeks the most suitable adaptation for the variations in the system context.

The adaptation process consists of detecting and storing variations in the system context (in the O_i). These variations start the adaptation process in which the first step is to decide which operations have to be executed on the architectural model to achieve the adaptation ($\mathcal{C}$). Once these actions have been identified, it proceeds to locate the right transformation rules to execute ($\mathcal{R}_\mathcal{R}$) from the rule repository (R_r). When these rules have been located and rated, they are selected ($\mathcal{R}_\mathcal{S}$) and a new model is generated with the rules to be executed (R_s). Then, the repository is updated, generating information on the history of the transformation rule use and updating their rates ($\mathcal{R}_\mathcal{L}$).

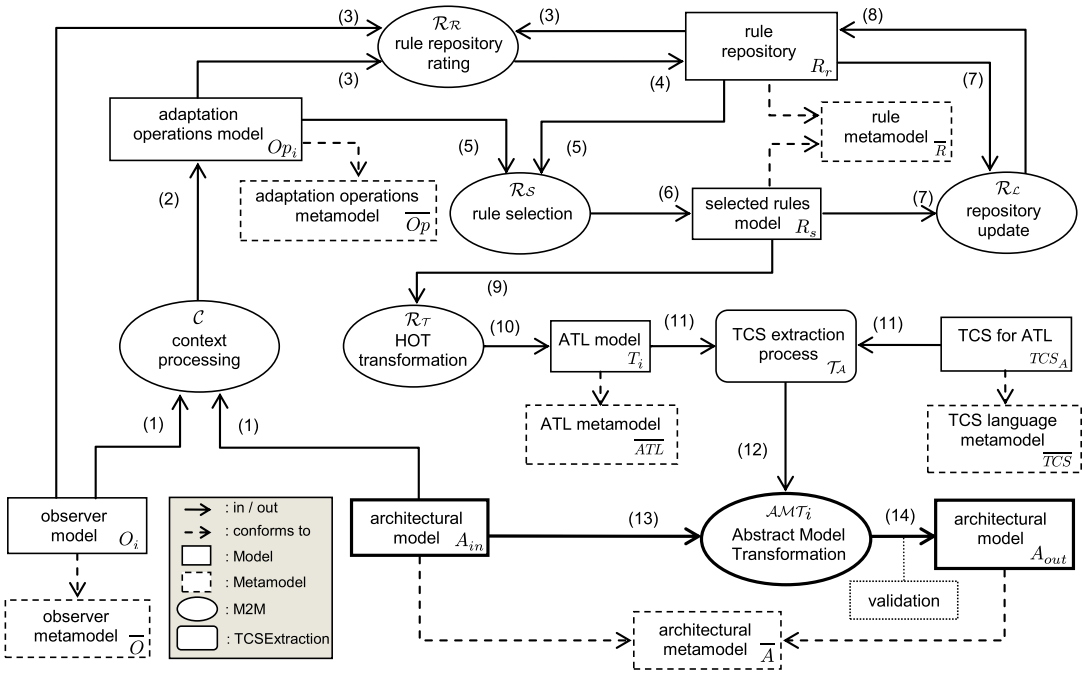


Figure 9. Adaptation schema.

By applying a HOT-type M2M transformation ($\mathcal{R}_{\mathcal{T}}$) followed by a textual concrete syntax (TCS) extraction ($\mathcal{T}_{\mathcal{A}}$), the ATLAS transformation language (ATL) transformation code called abstract model transformation (AMT) $\mathcal{AMT}_i$ is generated from the selected rules. This M2M process, which is not pre-set and has been performed dynamically, transforms the starting architectural model (A_{in}) into the adapted architectural model (A_{out}). This model transformation contains the transformation rules that were selected after processing the context information and according to the rules in the repository. The generation of the adapted architectural model carries an implicit validation of the model generated in the $\mathcal{AMT}_i$ transformation process.

3.2. Adaptation abstraction

The design of this process through different pieces, as well as the existence of common types of elements, resulted in the conceptualization of this process from a more abstract level. Thus, if the adaptation process is defined abstractly, it can be changed or improved by constructing a different transformation schema based on the metamodel that describes it. Furthermore, this 'modularization' provides it with certain flexibility when each model and transformation that participates in the process is developed. This metamodel (see Figure 10) was constructed using EMF. A `TransformationSchema` is comprised of metamodels, models, transformations, and TCS [54] extractions. TCS extractions represent processes that generate concrete textual syntax. The generic purpose of these processes is to generate text from reading a source model and another defining the correspondence between the first one and a textual syntax.

In our case, the `TCSExtractions` are used to generate the M2M transformation code from reading a `source` model in the adaptation schema in the transformation language defined by the `concrete_syntax` model. To be more precise, we use these processes to dynamically generate the code written in ATL [52] for the M2M transformations that adapt the architectural models based on the adaptation rules that have been selected for a given instant in the system. Each `Model` in the schema identifies and defines each model in the system, which is constructed according to its `Metamodel`. The `M2M` elements identify the M2M transformations that take place during adaptation. These transformations generate one or more output models based on one or more input models.

3.3. Adaptation schema definition

Starting from the previous metamodel, we are able to formally describe the process shown in Figure 9. Therefore, the adaptation schema (S_A) is defined in the following manner: $S_A = \{M, \overline{M}, \mathcal{M}, \mathcal{T}, I, O, C\}$. Thus, S_A is comprised of a set of models M, a set of metamodels $\overline{M}$, a set of M2M transformations $\mathcal{M}$, a set of TCS extractions $\mathcal{T}$, a set of input relationships I, a set of output relationships O, and a set of model-metamodel conformance relationships C. The set of metamodels $\overline{M}$ in the adaptation schema is defined as $\overline{M} = \{\overline{A}, \overline{O}, \overline{Op}, \overline{R}, \overline{ATL}, \overline{TCS}\}$, where $\overline{A}$ is the architectural metamodel, $\overline{O}$ the observer metamodel, $\overline{Op}$ the adaptation operations metamodel,

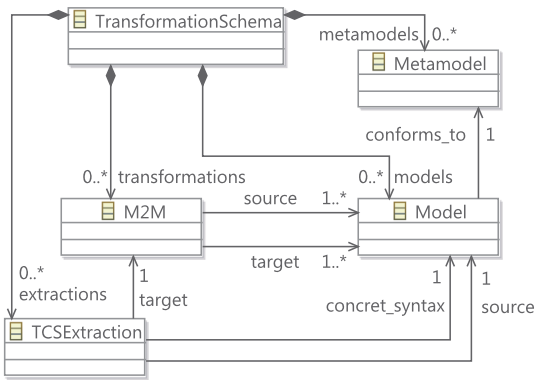


Figure 10. Adaptation process abstraction.

$\overline{R}$ the rule metamodel, $\overline{ATL}$ is the ATL metamodel, and $\overline{TCS}$ the TCS metamodel. The set of models at any given instant i is defined as $M = \{A_{in}, A_{out}, O_i, Op_i, R_r, R_s, T_i, TCS_A\}$, where A_{in} is the input architectural model, A_{out} is the output architectural model, O_i is the observer model at instant i, Op_i is the adaptation operations model, R_r the rule repository model, R_s the selected rule model, T_i the transformation model, and TCS_A the TCS model for the ATL language.

In the case of M2M transformations, $\mathcal{M}$ is defined as $\mathcal{M} = \{\mathcal{C}, \mathcal{R}_\mathcal{R}, \mathcal{R}_\mathcal{S}, \mathcal{R}_\mathcal{L}, \mathcal{R}_\mathcal{T}, \mathcal{AMT}_i\}$. $\mathcal{C}$ identifies the transformation process that calculates the adaptation operations that must be carried out (Op_i) by processing the context information provided by the input architectural model (A_{in}) and by the observer model (O_i). Process $\mathcal{R}_\mathcal{R}$ is in charge of rating the repository rules (R_r) based on operations selected (Op_i) and the properties of the context given by the observer model (O_i). $\mathcal{R}_\mathcal{S}$ represents the M2M process that generates the selected rule model (R_s) from rule repository (R_r) according to the operations indicated by Op_i. $\mathcal{R}_\mathcal{L}$ is the transformation process that updates the attributes of rule repository R_r for the following adaptation process based on the selected rules R_s. $\mathcal{R}_\mathcal{T}$ corresponds to the M2M transformation that generates the transformation model (T_i) from the selected rules R_s. Finally, $\mathcal{AMT}_i$ represents the transformation process that adapts the input architectural model A_{in} into output A_{out}. The TCS extractions in our adaptation schema are defined as $\mathcal{T} = \{\mathcal{T}_\mathcal{A}\}$. That is, our adaptation schema is comprised only of a TCS extraction called $\mathcal{T}_\mathcal{A}$, which reads the transformation model (T_i) and the TCS model for ATL (TCS_A) and generates the ATL code for the $\mathcal{AMT}_i$ transformation.

For the description of the set of input relationships I, an *input operator* $\xrightarrow{in}$, which is applied to elements in M, $\mathcal{M}$, and $\mathcal{T}$, is defined as follows: $I = \left\{m \xrightarrow{in} p \,/\, m \in M,\ (p \in \mathcal{M}) \vee (p \in \mathcal{T}),\ \mathcal{M} \cap \mathcal{T} = \emptyset\right\}$. This set describes all the relationships that originate in a model and have either an M2M transformation or TCS extraction process as their destination. Therefore, the set of input relationships I is comprised of the following: $\left\{O_i \xrightarrow{in} \mathcal{C},\ A_{in} \xrightarrow{in} \mathcal{C},\ O_i \xrightarrow{in} \mathcal{R}_\mathcal{R},\ Op_i \xrightarrow{in} \mathcal{R}_\mathcal{R},\ R_r \xrightarrow{in} \mathcal{R}_\mathcal{R},\ Op_i \xrightarrow{in} \mathcal{R}_\mathcal{S},\ R_s \xrightarrow{in} \mathcal{R}_\mathcal{L},\ R_r \xrightarrow{in} \mathcal{R}_\mathcal{L},\ R_s \xrightarrow{in} \mathcal{R}_\mathcal{T},\ T_i \xrightarrow{in} \mathcal{T},\ A_{in} \xrightarrow{in} \mathcal{AMT}_i\right\}$. For the description of the set of output relationships O, an *output operator* $\xrightarrow{out}$ that is also applied to the elements in M, $\mathcal{M}$, and $\mathcal{T}$ must be defined in the following manner: $O = \left\{p \xrightarrow{out} q \,/\, (p \in \mathcal{M} \wedge q \in M) \vee (p \in \mathcal{T} \wedge q \in \mathcal{M}), \mathcal{M} \cap \mathcal{T} = \emptyset\right\}$. Therefore, set O is comprised of those relationships that originate in an M2M transformation and have a model as their destination and of those that originate in a TCS extraction and have an M2M process as their destination, that is, $\left\{\mathcal{C} \xrightarrow{out} Op_i, \mathcal{R}_\mathcal{R} \xrightarrow{out} R_r, \mathcal{R}_\mathcal{S} \xrightarrow{out} R_s, \mathcal{R}_\mathcal{L} \xrightarrow{out} R_r, \mathcal{R}_\mathcal{T} \xrightarrow{out} T_i,\ \mathcal{T}_\mathcal{A} \xrightarrow{out} \mathcal{AMT}_i, \mathcal{AMT}_i \xrightarrow{out} A_{out}\right\}$. Finally, to describe set C, the conformance operator $\dashrightarrow$, which is applied between elements of sets M and $\overline{M}$, is defined as $C = \left\{m \dashrightarrow mm \,/\, m \in M,\ mm \in \overline{M}\right\}$ and represents the set of conformance relationships between the models (M) and metamodels ($\overline{M}$) that participate in the adaptation schema (S_A). Therefore, $C = \left\{A_{in} \dashrightarrow \overline{A}, A_{out} \dashrightarrow \overline{A},\ O_i \dashrightarrow \overline{O},\ Op_i \dashrightarrow \overline{Op},\right.$

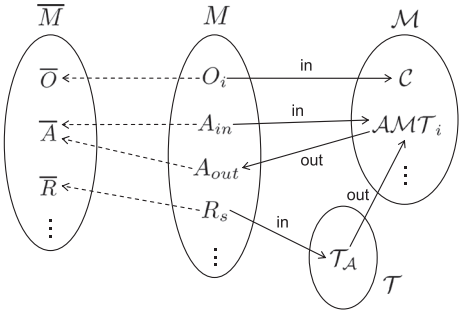


Figure 11. Adaptation schema operators.

$R_r \dashrightarrow \overline{R}$, $R_s \dashrightarrow \overline{R}$, $T_i \dashrightarrow \overline{ATL}$, $TCS_A \dashrightarrow \overline{TCS}\}$. Figure 11 shows the main sets of the adaptation schema, some examples of the set elements, and how they are related through the described operators.

Definition of the adaptation schema as a transformation sequence facilitates modularization of adaptation and its division into subprocesses with isolated behaviors and their independent management. In our adaptation schema, six subprocesses are identified: *context processing* ($\mathcal{C}$), *rule repository rating* ($\mathcal{R}_\mathcal{R}$), *rule selection* ($\mathcal{R}_\mathcal{S}$), *repository update* ($\mathcal{R}_\mathcal{L}$), *transformation of the rule model into an M2M process* ($\mathcal{R}_\mathcal{T} + \mathcal{T}_\mathcal{A}$), and *architectural model adaptation* ($\mathcal{AMT}_i$). This modularization provides certain flexibility for changing adaptation behavior or improving transformation, because transformation $\mathcal{A}$ can be changed into another, $\mathcal{A}'$, without varying the adaptation schema, as long as they are interchangeable. Two transformations are said to be interchangeable and are designated as $\mathcal{A} \equiv \mathcal{A}'$ when the same number and type of models are received as input and generated as output: $\mathcal{A} \equiv \mathcal{A}' \Leftrightarrow \forall \{x \to \mathcal{A}\} \in I, \forall \{\mathcal{A} \to y\} \in O, \exists \{z \to \mathcal{A}'\} \in I, \exists \{\mathcal{A}' \to w\} \in O \,/\, x = z, y = w \,;\, x, y, z, w \in M \,;\, \mathcal{A}, \mathcal{A}' \in \mathcal{M}$. This does not mean that these transformations are equivalent and have the same behavior, but it represents the ability of our proposal to exchange transformation modules within the adaptation schema. To make it easier to follow the elements in the adaptation schema, Table I shows the equivalence between the symbols defined and their meaning. This formal notation is used throughout the rest of the article to refer to each element of the adaptation schema and for the algorithms of the described subprocesses.

Table I. Equivalence of symbols in adaptation process.

Symbol	Description and alt. nomenclature	Symbol	Description and alt. nomenclature
M	Set of models	A_{out}	Output architectural model
$\overline{M}$	Set of metamodels	$\overline{O}$	Observer metamodel
$\mathcal{M}$	Set of M2M transformations	Op_i	Adaptation operation model
$\mathcal{T}$	Set of TCS extractions	R_r	Rule repository model
I	Set of input relationships	R_s	Selected rule model
O	Set of output relationships	T_i	Transformation model
C	Set of conformance relationships	TCS_A	TCS model for ATL language
$\overline{A}$	Architectural metamodel	$\mathcal{C}$	M2M transformation for processing context information
$\overline{O}$	Observer metamodel	$\mathcal{R}_\mathcal{R}$	M2M transformation for rule repository rating
$\overline{Op}$	Adaptation operation metamodel	$\mathcal{R}_\mathcal{S}$	M2M transformation for rule selection
$\overline{R}$	Rule metamodel	$\mathcal{R}_\mathcal{L}$	M2M transformation for updating the R_r values
$\overline{ATL}$	ATL language metamodel	$\mathcal{R}_\mathcal{T}$	HOT M2M tranformation for generating T_i
$\overline{TCS}$	TCS language metamodel	$\mathcal{AMT}_i$	M2M transformation for adapting architectural models
A_{in}	Input architectural model	$\mathcal{T}_\mathcal{A}$	TCS extraction for generating the $\mathcal{AMT}_i$ transformation

M2M, model-to-model; TCS, textual concrete syntax; ATL, ATLAS transformation language; HOT, higher-order transformation; AMT, abstract model transformation.

Table II. Summary of the process.

Subsection	Subprocess	In	Out
4.2	Processing the context ($\mathcal{C}$)	O_i, A_{in}	Op_i
4.3	Rating the rules ($\mathcal{R}_\mathcal{R}$)	O_i, Op_i, R_r	R_r
4.4	Selecting the rules ($\mathcal{R}_\mathcal{S}$)	Op_i, R_r	R_s
4.5	Updating the rule repository ($\mathcal{R}_\mathcal{L}$)	R_s, R_r	R_r
4.6	Transforming the rules ($\mathcal{R}_\mathcal{T}$)	R_s	T_i
	Extracting the rules ($\mathcal{T}_\mathcal{A}$)	T_i, TCS_A	$\mathcal{AMT}_i$

TCS, textual concrete syntax; AMT, abstract model transformation.

The following section describes in detail the adaptation methodology developed by using the adaptation process. With this aim, each subsection focuses on each of the subprocesses in the adaptation schema, as summarized in Table II.

4. ADAPTATION METHODOLOGY

Once the formal definition of the adaptation process has been presented, it is necessary to describe each of the subprocesses involved. For this purpose, first, we will set up an adaptation scenario and then we will explain the behavior of each module through its application in a common case study provided by this scenario.

4.1. Adaptation scenario

Regarding our application domain, our proposal is intended to adapt component-based GUIs. Therefore, some of the subprocesses in the adaptation schema explained in the succeeding text must be constructed specifically for this domain. In particular, the behavior of $\mathcal{C}$ and $\mathcal{R}_{\mathcal{R}}$ processes depend on the execution domain. The scenario is the following. A system user is interacting with a mashup GUI which contains the following components: an e-mail service (`Email`), a chat (`Chat`), an audio component (`Audio`), a low-quality video (`VideoLQ`), a high-quality video (`VideoHQ`), a file exchange component (`FileSharing`, e.g., Dropbox type), and a shared blackboard (`Blackboard`).

The system takes the following context status variables into account: available bandwidth (*bandwidth*), available free memory (*sysPerformance*), average size of shared files (*averageFileSize*), and the profile of the user interacting with the interface (*userProfile*). The *profile* of the user who is interacting with the interface affects the components that will be offered. In this scenario, a normal *user* will not have the available file sharing and blackboard components, in contrast to the other two profiles (*technician* and *politician*). Otherwise, the *technician* profile will not have the available high quality video component.

The adaptation process determines what changes must be made in the architecture configuration representing the UI depending on the values of the context variables. Thus, components are inserted or deleted according to the context status and the current state of the architecture. According to the scenario, the system manages a starting GUI having four components: `Email`, `Chat`, `Audio`, and `VideoLQ` (low-quality video). In this case, `GUI` is a *complex* component containing the aforementioned four *simple* components, and the transformation rules related to these components must incorporate some operations to manage this containment relationship. The corresponding architectural model is shown in Figure 12. Because changes in the context variables show that there has been an increase in bandwidth and available memory and that the user is sharing a large number of large-sized files over the chat component, the adaptation component will make a decision to reconfigure the UI by deleting the low-quality video and inserting a high-quality video component and a file sharing component.

Next, we will describe how this adaptation is achieved. In addition, the case study will use two different rule repositories so that the difference in the results can be observed. The two repositories illustrating the adaptation are called R_A as shown in Table III, and R_B, made up of rules from the

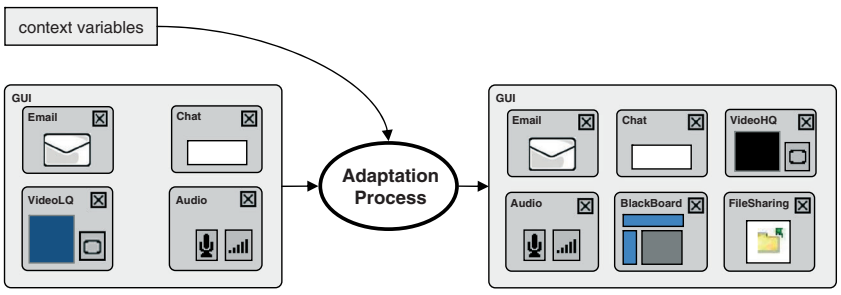


Figure 12. Adaptation scenario.

Table III. Rule repository model A (R_A).

Rule	Action	p	b	s	w
Insert_Email	InsertEmail	t	10	10	30
Insert_Chat	InsertChat	f	20	40	40
Insert_Audio	InsertAudio	f	30	50	21
Insert_Video1	InsertVideoHighQ	f	80	80	14
Insert_Video2	InsertVideoLowQ	f	40	60	15
Insert_Video3	InsertVideoHighQ	f	100	120	40
Insert_Blackboard1	InsertBlackboard	f	100	50	22
Insert_Blackboard2	InsertBlackboard	f	80	70	10
Insert_FileSharing	InsertFileSharing	f	300	100	9
Delete_Email	DeleteEmail	t	—	—	30
Delete_Chat	DeleteChat	t	—	—	30
Delete_Audio	DeleteAudio	t	—	—	30
Delete_VideoHighQ	DeleteVideoHighQ	t	—	—	30
Delete_VideoLowQ	DeleteVideoLowQ	t	—	—	30
Delete_Blackboard	DeleteBlackboard	t	—	—	30
Delete_FileSharing	DeleteFileSharing	t	—	—	30

Table IV. Extra rules in rule repository model B (R_B).

Rule	Action	p	b	s	w
Insert_Recorder	InsertVideoHighQ	t	200	200	0
Connect_RVHQ	InsertVideoHighQ	t	—	—	0
Delete_Recorder	DeleteVideoHQ	t	—	—	0
Delete_Connect_RVHQ	DeleteVideoHighQ	t	—	—	0

first repository adding the rules shown in Table IV. The headings *p*, *b*, *s*, and *w* in the tables represent the *is_priority*, *bandwidth*, *sysPerformance*, and *weight* attributes, respectively.

In order to illustrate the rule actions on the tables, we take as an example the rules `Insert_Video1`, `Insert_Video3`, `Insert_Recorder`, and `Connect_RVHQ`. These rules are responsible, respectively, for inserting a high quality video component, inserting a recording component, and connecting the ports of both components. The reason why the four rules are related with the same action (*InsertVideoHighQ*) is because that is precisely the mechanism (adaptation polymorphism) that our proposal provides to select multiple adaptation rules associated with the same adaptation operation (see Section 4.4).

4.2. Processing the context

The first step toward adaptation according to our proposal (Figure 9) is to decide what actions must be executed depending on the system context status and the current state of the system architecture. That is, the system must adapt to any variation in its context, modifying the configuration of its architecture, if necessary. At any given instant ($t = i$), the system context status is reflected in the observer model (O_i), and the current state of the architecture is determined by the architectural model (A_{in}). Thus, the context processing is done by an M2M transformation called $\mathcal{C}$; this uses input from the observer model O_i and the architectural model A_{in} to generate the operations that must be executed to adapt the starting architectural model according to the variations in the system context, following a certain logic defined a priori. These operations are stored in the architectural adaptation operations model (Op_i).

As a practical validation of our adaptation proposal, we have selected the component-based UI domain. We have therefore implemented the $\mathcal{C}$ model transformation with the decision logic of operations to be applied to that concrete domain with some specific preconditions. Table V summarizes this logic. The first column describes the variables in the system environment: the bandwidth (*bandwidth*, *b*) and the main memory (*sysPerformance*, *s*) available to the system, in addition to the size of the files shared (*averageFileSize*, *a*). The values of these variables determine the adap-

Table V. Processing the context ($\mathcal{C}$).

System context state	u	Components
$b \in [0, bandChat())$ $s \in [0, sysChat())$	U, T, P	E
$b \in [bandChat(), bandAudio())$ $s \in [sysChat(), sysAudio())$	U, T, P	E, C
$b \in [bandAudio(), bandVideoL())$ $s \in [sysAudio(), sysVideoL())$	U, T, P	E, C, A
$b \in [bandVideoL(), bandFileShar())$ $s \in [sysVideoL(), sysFileShar())$ $a < sizeFileSharing()$	U, T, P	E, C, A, VL
$b \in [bandFileShar(), bandVideoH())$	U	E, C, A, VL
$s \in [sysFileShar(), sysVideoH())$	T	E, C, A, VL, F
$a \geqslant sizeFileSharing()$	P	E, C, A, VL, F
$b \in [bandVideoH(), bandBlackb())$	U	E, C, A, VH
$s \in [sysVideoH(), sysBlackb())$	T	E, C, A, VL
$a < sizeFileSharing()$	P	E, C, A, VH
$b \in [bandVideoH(), bandBlackb())$	U	E, C, A, VH
$s \in [sysVideoH(), sysBlackb())$	T	E, C, A, VL, F
$a \geqslant sizeFileSharing()$	P	E, C, A, VH, F
$b \geqslant bandBlackb()$	U	E, C, A, VH
$s \geqslant sysBlackb()$	T	E, C, A, VL, B
$a < sizeFileSharing()$	P	E, C, A, VH, B
$b \geqslant bandBlackboard()$	U	E, C, A, VH
$s \geqslant sysBlackboard()$	T	E, C, A, VL, F, B
$a \geqslant sizeFileSharing()$	P	E, C, A, VH, F, B

```
1: if (components contains 'VideoHQ') = true then
2:     if (A_in.components contains 'VideoLQ') = true then
3:         Op_i.addOperation('DeleteVideoLowQ')
4:     end if
5:     if (A_in.components contains 'VideoHQ') = false then
6:         Op_i.addOperation('InsertVideoHighQ')
7:     end if
8: end if
```

Figure 13. Example of selecting adaptation operations algorithm.

tation operations to be performed to maximize system efficiency. For this, thresholds have been defined using ATL *helpers* that represent the resources that require the components to work properly. For example, `bandChat()` represents the bandwidth necessary for the chat component, and `sysBlackboard()` represents the available main memory that the blackboard component requires. In addition, `sizeFileSharing()` is the file size threshold that must be exceeded for the file sharing component to be inserted. Thus, the action or actions to be executed are determined by whether or not the values of context variables are within the intervals defined by these thresholds.

The second column describes context information related to the profile of the user who is interacting with the architecture. Depending on his knowledge and interaction with the system, this variable, (*userProfile*, *u*) may be user (a U), technician (T), or politician (P). The third column shows the *components* necessary for adaptation, which may be Email (E), chat (C), audio (A), low quality video (VL), high quality video (VH), file sharing (F), and blackboard (B). The logic must take into account components that are already in the starting architectural model A_{in} (Figure 12) to determine the adaptation operations that must be executed by the adaptation process, as shown in Figure 13. This fragment of the process shows a case in which high-quality video must be inserted. If there is already a low-quality video component in the architecture, it must be deleted (lines 2 to 4), and if the high-quality video component is already in the architecture, the insertion operation is not included (lines 5 to 7).

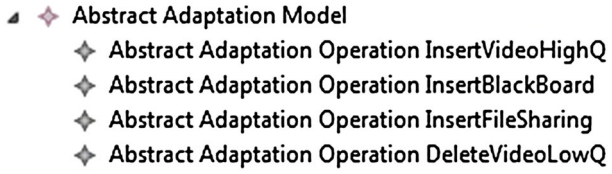


Figure 14. Adaptation operations.

It should be mentioned that Table V shows the simplified process logic, and therefore, there are combinations of context variables that are not shown. For example, one of the variables meets the condition for interval ($b \in [bandAudio(), bandVideoL())$), whereas the other one does not ($s \notin [sysAudio(), sysVideoL())$). If the variable that does not meet the condition is over the threshold for the interval, the operations selected are the same as for the affirmative case. If, on the contrary, it is below it, the operations selected are those that meet the condition immediately below the values in the table, keeping in mind the restriction just explained. To summarize, the purpose of $\mathcal{C}$ transformation is to select the adaptation operations that make the system most efficient for the state of its context. This is accomplished by offering the right components depending on their use of available resources and the defined profile for adapting the system to the context. The complete source code of the $\mathcal{C}$ transformation process is available on our website [47] (in the *Sources* section).

Focusing on the case study scenario, let us suppose that circumstances exogenous (state of context) to the system are reflected in the observer model (O_i) in the following manner: the available bandwidth (*bandwidth*) is 2000 Kbps, the main memory available (*sysPerformance*) is 4000 MB, the average file being shared (*averageFileSize*) is 300 MB, and the *profile* of the user interacting with the interface is 'politician'. We also assume the following thresholds for the *helpers* used in $\mathcal{C}$: bandBlackboard() = 1750 (Kbps), sysBlackboard() = 3500 (MB), and sizeFileSharing() = 200 (MB). Keeping in mind the context variable values and the initial architectural model, the operations model to be performed Op_i will have the values shown in Figure 14. Following the $\mathcal{C}$ logic, we find that $bandwidth >= 1750$ (Kbps), $sysPerformance >= 3500$ (MB), and $averageFileSize >= 200$ (MB), and because the `Email`, `Chat`, and `Audio` components are already in the architecture, it is unnecessary to reinsert them. And as the `VideoHQ` component is among those selected by the logic, this implies deleting the `VideoLQ` component (operation *DeleteVideoLowQ*) in the initial architectural model (A_{in}).

4.3. Rating the rules

According to our proposal, the rules in the rule repository (R_r) must be rated at run-time so that the rules with a higher numerical value in a certain attribute will be those run in later processes. This transformation rule attribute is called the *mark*. The M2M transformation in charge of this rate is called $\mathcal{R}_{\mathcal{R}}$, whose source code is available on the aforementioned website [47]. The input to this process is the architectural adaptation operations model (Op_i), the transformation rule repository model (R_r), and the observer model (O_i), as shown in Figure 9, steps (3) and (4).

The Op_i model is used to indicate the adaptation operations that have been calculated by $\mathcal{C}$ as explained in Section 4.2. The adaptation operation associated with each of the rules and a set of context variables on which that rule has a repercussion are defined in the R_r. The values of this set are used to calculate the rule's *mark*. Each rule is rated independently and it is worth mentioning that in our proposal, we do not rate the possible configurations of the architecture, but we evaluate each repository rule (which acts on the components of the architecture). For example, the rule `InsertAudio` defines two variables in this set $bandwidh = 30$ and $sysPerformance = 50$, which mean that the selection of this rule implies the use of a 30-Kbps bandwidth and 50-MB main system memory, because as a result of its inclusion, an audio component is added. O_i contains the information on which context variables participate in the execution of the adaptation process.

As output, the $\mathcal{R}_{\mathcal{R}}$ transformation modifies the rules in the R_r rule model using a metric that rates the rules with action coinciding with any of the operations included in operations model Op_i. This rate is reflected in the rule attribute *mark*. The attribute *mark* represents the value of a simplified

Table VI. Rating rule repository model A.

Rule	Action	b	s	mark
Insert_Video1	InsertVideoHighQ	80	80	0.00625
Insert_Video3	InsertVideoHighQ	100	120	0.00455
Insert_Blackboard1	InsertBlackboard	100	50	0.00667
Insert_Blackboard2	InsertBlackboard	80	70	0.00667
Insert_FileSharing	InsertFileSharing	300	100	0.0025
Delete_VideoLowQ	DeleteVideoLowQ	—	—	0.0

Table VII. Rating rule repository model B.

Rule	Action	b	s	Mark
Insert_Video1	InsertVideoHighQ	80	80	0.00625
Insert_Video3	InsertVideoHighQ	100	120	0.00455
Insert_Recorder	InsertVideoHighQ	200	200	0.0025
Connect_RVHQ	InsertVideoHighQ	—	—	0.0
Insert_Blackboard1	InsertBlackboard	100	50	0.00667
Insert_Blackboard2	InsertBlackboard	80	70	0.00667
Insert_FileSharing	InsertFileSharing	300	100	0.0025
Delete_VideoLowQ	DeleteVideoLowQ	—	—	0.0

utility function which maximizes system performance for the context variables at any given instant, in our case study, by minimizing the use of both the bandwidth (*bandwidth*) and the main memory (*sysPerformance*). Therefore, the value of the score ($mark_r$) calculated for a rule whose adaptation operation coincides with any of those present in the Op_i is inversely proportional to the sum of the attributes *bandwidth* and *sysPerformance*: $mark_r = 1 \ / \ (r.bandwidth + r.sysPerformance)$.

This formula is implemented as part of the model transformation and it is written in ATL code. This code and the complete $\mathcal{R}_\mathcal{R}$ process are available on the mentioned web page with the M2M transformation sources. It is true that $\mathcal{C}$ and $\mathcal{R}_\mathcal{R}$ may be executed together in a single transformation. The purpose of separating them is to make it easier for modification of the adaptation logic to distinguish between (i) calculation of the target adaptation operations and (ii) criteria for calculating the rule repository mark. We are aware that the method of calculating the scores of the rules is simple, but it is an example of fitness function to evaluate the adaptation rules. On the other hand, it is important to note that the dependencies between components also affect non-functional properties, but this is already covered in the values of the rules that are related to the context variables, which are set at design-time by an expert.

Applying this scoring process to our case study, the rule repositories shown in Tables VI and VII are obtained. There are shown only the rules that have been affected by the process. That is, those whose *action* coincides with any of the adaptation operations in the adaptation operations model Op_i. For example, the *mark* in `Insert_Video1` is the result of the operation $1/(80 + 80) = 0.00625$, whereas the mark of the rule `Connect_RVHQ` is 0.0, because that rule does not have the goal to add a component that influences the use of system resources.

4.4. Selecting the rules

After the rules in the rule repository model R_r have been rated, continuing the adaptation schema in our proposal (Figure 9, steps (5) and (6)), the next step for run-time adaptation is to select the right subset of rules. The M2M transformation process in charge of making this selection is called $\mathcal{R}_\mathcal{S}$. Its input is the transformation rule repository model R_r and the adaptation operations model Op_i, and it generates the selected transformation rules model (R_s). The pseudocode for this process can be seen in Figure 15, and the corresponding ATL code can be found on the web page [47].

This process filters the rule repository, so it starts from the complete rule set (line 1) and removes those rules that will not be selected (lines 4, 8, and 9). This algorithm deletes the transformation rules in the R_r rule repository whose attribute *action* does not coincide with any of the operations

```
process R_S
 1: R_s = R_r
 2: for n = 1 → card(R_r) do
 3:     if (Op_i contains R_r[n].action) = false then
 4:         R_s.deleteRule(R_r[n])
 5:     else
 6:         if R_r[n].is_priority = false then
 7:             if isBiggestMark(R_r[n]) = false then
 8:                 R_s.deleteRule(R_r[n])
 9:             else
10:                 if isBiggestWeight(R_r[n]) = false then
11:                     R_s.deleteRule(R_r[n])
12:                 end if
13:             end if
14:         end if
15:     end if
16: end for
17: for n = 1 → card(R_s) do
18:     if hasConflict(R_s[n]) = true then
19:         R_s.deleteRule(R_s[n])
20:     end if
21: end for
```

Figure 15. Rule selection algorithm ($\mathcal{R}_\mathcal{S}$ process).

included in the adaptation operations model Op_i (lines 3 and 4). Those rules marked as priorities ($is\_priority = true$) are always selected (line 6). The one with the highest *mark* is selected from among those that are non-priority (lines 7 and 8), because it is the one that best meets the requirements for the adaptation operation to be performed. If two or more rules meet the requirement and have the same *mark*, the one with the highest *weight* is chosen (lines 10 and 11), because it is the one that has been used the most, which shows that, a priori, it is the one that adapts the most to the operation to be performed and to the system requirements. Further details on updating rule weight are given in Section 4.5. The function $isBiggestWeight(R_r[n])$ is in charge of calculating if this rule is the one with the biggest *weight* among all with the same *action*, as $isBiggestMark(R_r[n])$ does with regard to the *mark* attribute. The following excerpt shows the corresponding ATL code.

```
helper def: isBiggestWeight(checked_rule : RMM!Rule) : Boolean = not(RMM!Rule ->
  allInstances() -> select(r | (r.rule_name <> checked_rule.rule_name) and
  (r.action = checked_rule.action) and (r.mark = checked_rule.mark)) ->
  exists(r | r.weight > checked_rule.weight));
```

The selection loop does not ensure that there are no rules which could be applied to the same element of the input architectural model. This means that there could be some conflicts between the selected rules. In ATL, as in other transformation languages with declarative behavior, it will cause a run-time error when the transformation tries to apply two (or more) different rules to the same element. Therefore, our selection process executes a filter loop to check and remove the conflictive rules (lines 17 to 21 of Figure 15). The criteria implemented for resolving these conflicts are selected to the rule with the shortest rule name. More implementation details about this filter process are also available on the web containing the M2M transformation sources.

In the proposed scenario, the selected rule model (R_s) generated is different depending on which repository is used. If R_A is used (i.e., $R_r = R_A$), the list of rules selected is `Insert_Video3`, `Insert_Blackboard1`, `Insert_FileSharing`, and `Delete_VideoLowQ`. The first rule is selected because it is the rule with *action* equal to *InsertVideoHighQ*, which has the highest rating. The second rule is selected because, although the two rules whose *action* is *InsertBlackboard* have the same rating, the one selected is the one that has the highest *weight*. The other two rules are selected because they are the only ones in the repository with that *action*.

If the repository R_B (i.e., $R_r = R_B$) is used, the rules selected are the following: `Insert_Video3`, `Insert_Blackboard1`, `Insert_FileSharing`, `Delete_VideoLQ`, `Insert_Recorder`, and `Connect_RVHQ`. The reason for selecting the first four rules is the same as for R_A. The last two rules in the list (`Insert_Recorder` and `Connect_RVHQ`)) are selected because their *action* is listed in the adaptation operations model (Op_i), and although they do not have the highest rating, their *is_priority* attribute is `true`, so they are selected.

4.5. Updating the rule repository

In our proposal, the transformation rules in the rule repository R_r must be modified such that the use frequency of the rule (during all system executions) is recorded. This numerical value is reflected by the rule attribute *weight*, because, as explained in Section 4.4, it may be necessary for $\mathcal{R}_\mathcal{S}$ to use this attribute to generate the right adaptation rule subset R_s. The M2M model transformation process in charge of this task is called $\mathcal{R}_\mathcal{L}$. The input to this process is the transformation rule repository R_r and the selected rule model R_s generated by $\mathcal{R}_\mathcal{S}$, and it updates the transformation rule repository model using the weight attribute of the transformation rules modified according to their use in the adaptation stage at instant i (Figure 9, steps (7) and (8)).

The logic for updating the repository rules is explained as follows and shown as pseudocode in Figure 16. Otherwise, the corresponding ATL code is available on the web [47]. The run counter (*run_counter*) of the rules, whose *action* coincides with any *action* of the rules that have been selected, is increased whether or not it was selected (lines 3 and 9 of Figure 16). In the R_r, if the repository rules coincide with the transformation rules (R_s) generated by $\mathcal{R}_\mathcal{S}$, the attribute that shows that the rule has been used for adaptation (*selection_counter*) is updated (line 4). The use frequency (*ratio*) is updated for all the rules in which any counter has been changed (lines 5 and 10). The *weight* of the repository rules that coincides with the rules selected is updated by applying a bonus based on the *ratio* (line 6). The weight of those whose *actions* coincide with any of those selected but were not selected themselves is decreased (line 11). The coefficients of *bonus* and *penalty* are set a priori and are used to influence the weight of the rule positively or negatively, depending on whether or not it is among those selected. The *mark* of the rules is reset to zero for the next adaptation process (line 14).

In the research and development of an algorithm for updating the rules after each step of the adaptation process, we tried to implement a weight-based mechanism which allows us to establish and modify the weight associated with a rule. This attribute would indicate the 'importance' of a rule compared with the other rules of the same type. Then, these weights could be used to influence the selection. Building on multiplicative weight methods [55], the rules can be selected randomly with probability proportional to the weights. We rely on this type of work to incorporate the weight attribute with the goal of having a mechanism to be able to perform different selection processes.

```
process R_L
 1: for n = 1 → card(R_r) do
 2:     if R_s contains R_r[n] then
 3:         R_r[n].run_counter ← R_r[n].run_counter + 1
 4:         R_r[n].selection_counter ← R_r[n].selection_counter + 1
 5:         R_r[n].ratio ← R_r[n].selection_counter/R_r[n].run_counter
 6:         R_r[n].weight ← R_r[n].weight + (1 + R_r[n].ratio) * R_r.bonus
 7:     else
 8:         if R_s.action = R_r[n].action then
 9:             R_r[n].run_counter ← R_r[n].run_counter + 1
10:             R_r[n].ratio ← R_r[n].selection_counter/R_r[n].run_counter
11:             R_r[n].weight ← R_r[n].weight − (1 + R_r[n].ratio) * R_r.penalty
12:         end if
13:     end if
14:     R_r[n].mark = 0
15: end for
```

Figure 16. Updating rule repository algorithm ($\mathcal{R}_\mathcal{L}$ process).

Table VIII. Updating R_A and R_B.

R_r	Rule	Weight	Ratio	Run_c	Selec_c
A and B	Insert_Video1	14→11	0→0.5	0→1	0
A and B	Insert_Video3	40→44	0→1.0	0→1	0→1
B	Insert_Recorder	0→4	0→1.0	0→1	0→1
B	Connect_RVHQ	0→4	0→1.0	0→1	0→1
A and B	Insert_Blackboard1	22→26	0→1.0	0→1	0→1
A and B	Insert_Blackboard2	10→7	0→0.5	0→1	0
A and B	Insert_FileSharing	9→13	0→1.0	0→1	0→1
A and B	Delete_VideoLowQ	30→34	0→1.0	0→1	0→1

In our case, the weight is used to select between rules which have the same mark. As these methods state, there are many options to update the weight value in order to be able to modify the selection operation. For example, the expert could manually modify them, or it is possible to develop an automatic process based on the impact caused by the selection of a specific rule or based on the use frequency. We decide to update the rules by storing some information about the use frequency and also to modify their weight from this data. We utilize bonus and penalty coefficients to enhance the speed with which the weights of the rules are changed. Moreover, these coefficients do not multiply directly the value of the ratio and we add a unit, because values between zero and one would reduce the impact of the variation. When the $\mathcal{R}_{\mathcal{L}}$ transformation process is applied to our case study scenario, we obtain the updated values for both repositories R_A and R_B, which are shown in Table VIII.

4.6. Transforming the selected rules into the $\mathcal{AMT}_i$

When the subset of rules selected has been found for the context conditions indicated by the observer model, according to the initial architectural model, the subprocess in charge of translating the selected rule model into ATL code is run. This code makes up the M2M transformation named as $\mathcal{AMT}_i$ (from AMT), which adapts the starting architectural model A_{in} into the adapted architectural model A_{out} (see Figure 9). This translation is done in two steps.

1. The M2M transformation called $\mathcal{R}_{\mathcal{T}}$, which is in charge of translating the selected rule model into the ATL transformation model, is run.
2. The TCS extraction called $\mathcal{T}_{\mathcal{A}}$, in charge of generating the ATL code from the transformation model obtained in the last step, is run.

The $\mathcal{R}_{\mathcal{T}}$ process is a HOT-type M2M transformation, because its input is the selected rule model R_s and it generates a transformation model T_i as output, which is constructed according to the ATL definition language ($\overline{ATL}$). The rule metamodel $\overline{R}$, with which both the selected rule models and the rule repository are defined, is a simplification of the ATL metamodel. Thus, the transformation rules that are stored in the repository may be constructed more conveniently than if the ATL metamodel was used. So, their manipulation (insertion or modification of rules) is easier, and furthermore, attributes associated with the rules can be defined (such as the *mark*, *weight*, and *action* attributes). These attributes provide additional information that is used by the context processing logic for rating and selecting the rules and so on and in general, information used by the adaptation process.

The complete code of the HOT process is available on the web [47]. Next, we present an excerpt from the code which shows an example of the $\mathcal{R}_{\mathcal{T}}$ process in charge of translating the deletion rules, defined according to the $\overline{R}$ rule metamodel, into ATL rules. This rule is run for rules that have to be run in refining mode (`is_refining = true`), whose target element is called '`drop`'. This target name converts it into a deletion rule. In addition, the ATL refining mode [56] allows transformation rules to be defined that affect only those elements which change from the source model to the destination.

```
rule CalledRuleDrop{
 from rm: RMM!CalledRule(
  rm.is_refining = true and rm.to_elements
    ->first().element_name = 'drop'
  )
 to
  atl: ATLMM!CalledRule(
   name <- rm.rule_name,
   outPattern <- out_pattern
  ),
  out_pattern: ATLMM!OutPattern(
   dropPattern <- drop_element
  ),
  drop_element: ATLMM!DropPattern
}
```

Then the $\mathcal{T}_{\mathcal{A}}$ process generates the ATL code with the transformation rules in charge of adapting the architectural model. This process is executed as a TCS extraction[§], which takes as input the transformation model found in the previous step and the TCS model which relates each element in the transformation model to its corresponding ATL code (see the section *Sources* in the web page [47]). For example, the following excerpt from the code shows two rules. The function of the first rule is to delete the existing low-quality video component (`VideoLQ`) from the UI. The second rule is responsible for connecting the `Recorder` and the `VideoHQ` components from binding its ports.

```
rule DeleteVideoLowQ{
 from
  f : AMM!SimpleAbstractComponent(
       f.component_name = 'VideoLQ'
      )
 to
  drop
}

rule ConnectRVHQ{
 from
  f : AMM!SimpleAbstractComponent(
       f.component_name = 'Recorder'
      )
 to
  t : AMM!SimpleAbstractComponent(
    component_name <- f.component_name,
    component_parent <- f.component_parent,
    ports <- f.ports
  ),
  c : AMM!Connector(
    connector_parent <- f.component_parent,
    end_A <- (thisModule.getSimpleComponent('Recorder').ports)->
      select(p | p.port_name = 'Recorder_video')->first(),
    end_B <- (thisModule.getSimpleComponent('VideoHighQ').ports)->
      select(p | p.port_name = 'VideoHighQ_video ')->first()
  )
}
```

[§]TCS Extractor library—org.eclipse.m2m.atl.drivers.emf4atl.tcs.extractor.TCSExtractor.

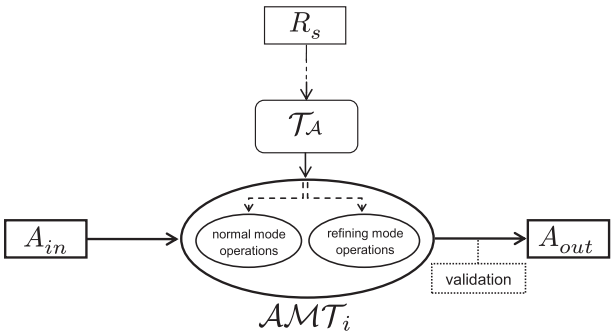


Figure 17. Abstract model transformation.

The code of two ATL model transformation processes that are executed sequentially is generated. The former transformation contains the rules that are executed in the 'normal mode' of ATL semantics; on the other hand, the latter is composed of the rules that are executed in 'refining mode' (Figure 17). The reason for separating the two kinds of rules is that they affect different sets of the input model: 'normal mode' rules are used to generate each of the expected target model elements, and 'refining mode' rules affect only those elements which change from the source model to the destination [56]. In our case, model transformations are used to adapt component-based system and, for example, a rule for deleting a component must necessarily be executed in refining mode, whereas a rule that inserts a new component must be defined to run in normal mode.

In order to get this distinction, our repository rules have an attribute indicating if the rule must be executed in refining mode or not and, consequently, $\mathcal{R}_\mathcal{T}$ and $\mathcal{T}_\mathcal{A}$ processes will generate the two model transformations that together make up the $\mathcal{AMT}_i$. As a consequence, our proposal relies on this particular execution mode of ATL and it cannot operate properly without it. Without this mode, the final transformation that handles the refactorization and adaptation of the model would have to rebuild from scratch all the elements by modifying only the parts that are changing. This option would imply worse execution times, and therefore, it has been discarded.

Otherwise, the selected rules (R_s) always contain a subset of helpers rules that allows the model transformation to check (using OCL operations) that the resulting elements in the target model have been properly managed and generated. Next, we will show two examples.

```
helper def: getSimpleComponent(c_name : String) : AMM!SimpleAbstractComponent =
 AMM!SimpleAbstractComponent-> allInstances()->select(c|c.component_name =
 c_name)->first();

helper def: checkRequiredPorts() : Boolean =
 AMM!Required->allInstances()->forAll(p | p.connector_parent.oclIsUndefined()
 = false);
```

The previous excerpt from the code shows two rules. The first one is in charge of getting a component from its name, and the goal of the second rule is to check that all required ports have a connector with another component of the model, that is, the dependencies of all components are solved by the architecture. An example of an $\mathcal{AMT}_i$ process can be found on the web page [47].

4.7. Obtaining the adapted architectural model

The last step in the process is to run the abstract model transformation ($\mathcal{AMT}_i$), which has been dynamically constructed and is in charge of adapting the architectural model that represents the UI. Figure 18 shows the result of applying our adaptation process by using the two repositories described in the case study.

Therefore, if R_A is used, the result is that the low-quality video component is replaced by a high-quality video component, and the blackboard and file-sharing components are inserted. On the

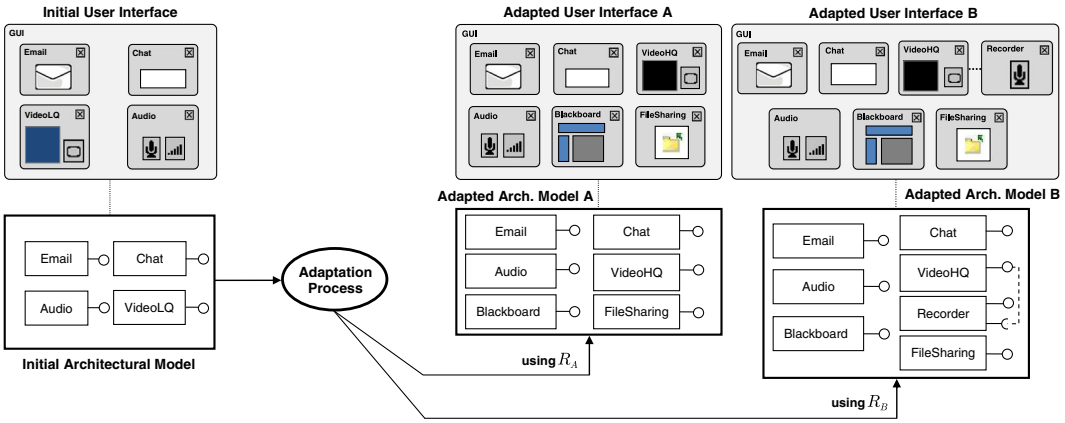


Figure 18. Adaptation variability.

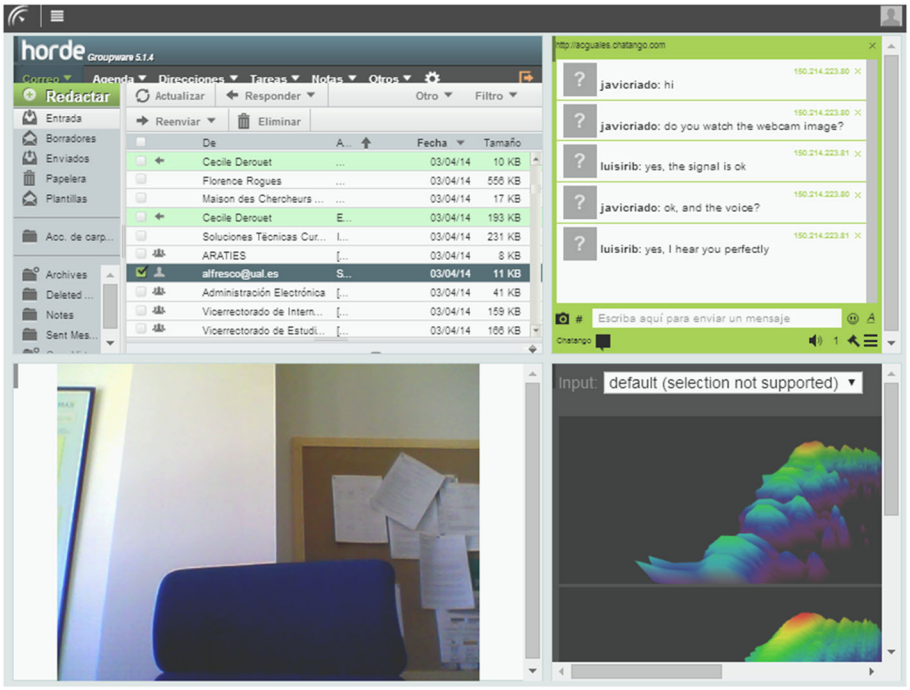


Figure 19. Mashup GUI example.

contrary, if R_B is used, the video component insertion is accompanied by the addition of a recording component which is related to the first one. This relationship is described in the architectural model through a connection between their interfaces and it is also represented by a link in the adapted UI (Figure 18).

As we have stated, the $\mathcal{AMT}_i$ transformation generated by the adaptation process contains a subset of helpers rules that ensure that the resulting elements in the target model have been properly generated. Furthermore, we assume that the initial model has been constructed correctly, and the rules in the repository perform consistent actions. Therefore, we can be sure that the adapted architectural models will be generated conforming the architectural metamodel. Nevertheless, at the end of the adaptation process, the system performs a validation of the resulting model. This validation checks that the OCL constraints and the structural definitions specified in the metamodel are fulfilled.

Figure 19 shows an example of the final GUI. It is the real UI application that is generated from the abstract representation of an architectural model, and it corresponds to the initial UI of Figure 18. We show this GUI for illustrative purposes, because this paper does not describe the realization process from which the final GUI is generated. Nevertheless, it is important to observe an example of the software obtained at the end of each step of the adaptation process. This GUI is available on the web [47].

5. IMPLEMENTATION AND ASSESSMENT

In this section, we describe the implementation developed with the aim of validating our approach and the experiments performed to evaluate the execution times. Moreover, the benefits and shortcomings of our proposal are discussed.

5.1. Validation and evaluation

To *validate* our proposal, we have developed a tool that enables us to execute the adaptation process on the architectural models that take part in the described case study scenario. It was implemented by using the Eclipse EMF, ATL, and TCS libraries and was deployed in a three-tier server architecture so that it can be tested from any platform without having to install any application or plugin. In this sense, a GUI is deployed in an Apache (httpd.apache.org) server to perform the functions of the tool's front-end (Figure 20). Otherwise, a Tomcat (tomcat.apache.org) server deploys the

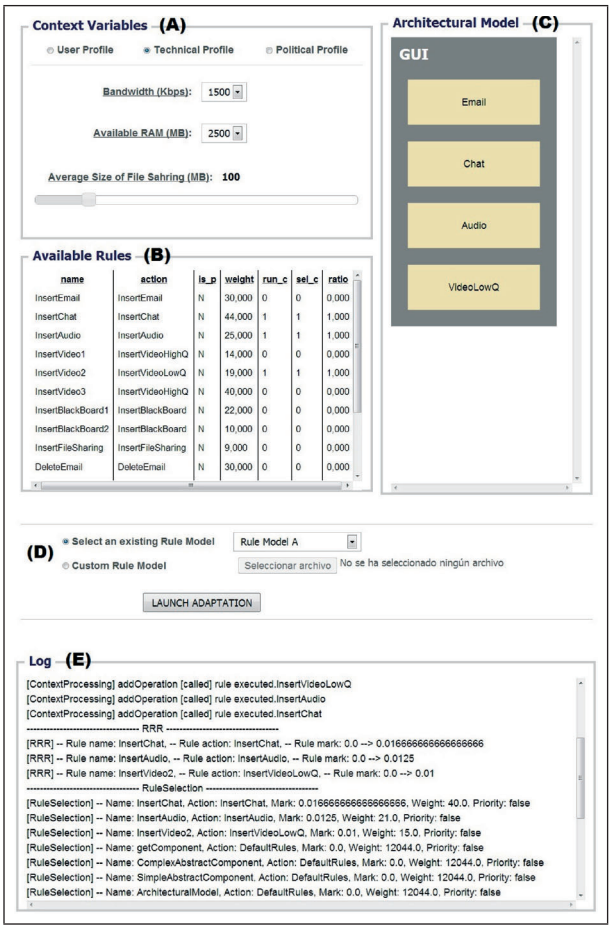


Figure 20. Validation tool.

ATL and TCS libraries and offers the model transformation services. In addition, another Tomcat server deploys the EMF libraries to provide model validation services. More information about the adaptation scenario and the tool is available in the *adaptation tool* section of the web page [47].

This tool (through the mentioned front-end) enables us to test input of different context variable values (A) and see how this variation affects the architectural model that is obtained at the end of the adaptation process (C). In addition, the tool shows information on the repository rules that are being used by the adaptation process (B). Furthermore, it makes it possible to select the rule repository that is going to be used from a series of predefined models or else provide our own repository of transformation rule models (D). When the adaptation process has been started, log information on the rules, run in the M2M transformations of the adaptation schema, is shown at the bottom of the tool (E). This information provides traceability, in addition to checking that the process is working properly.

To test and *evaluate* that the proposed adaptation process meets assumable run-times for our application domain (adaptation of component-based UIs), an experiment was developed. The experiment was run within the Eclipse `helios-SR2` *framework* in a machine with a 3.33 GHz Intel(R) Core(TM) i5 processor and 4 GB main memory. About 1000 iterations of the adaptation process were run in which the context variable values were changed at random. Execution time of each subprocess in the adaptation schema was measured, and the set of iterations was averaged in `ms` for each iteration. The experiment was done for different sizes of input models in the adaptation process: the rule repository model (R_r), the observer model (O_i), and the initial architectural model (A_{in}). In order to create these models, we randomly generated the required elements before launching each simulation of the experiment.

When we refer to random generation, it means that we implemented a test process that generates random values between all the possible values that the elements could have. This operation has been performed making use of the Java model code generated from the EMF metamodels [43]. Therefore, the models conform to the corresponding metamodel and contain valid values. Because of this, we do not need to accomplish any special operation to perform test oracles. In the case of models with 'extra' elements (i.e., elements which are created to increase the size of the input models), we generate model instances with variables or attributes that are not changing the adaptation behavior but are taken into account in the process algorithms and allow us to evaluate their performance. The evaluation process does not only take into account models with the size of the presented case study, this is because, now, we manage some example scenarios, with few context variables, not many components in the architecture and small rule repositories but there could be some other scenarios with a lot of context variables, a large amount of components and bigger rule repositories.

Figure 21 shows the results of the experiment using models from 100 to 10,000 elements. As described in the legend of the figure, the normal line shows the times when we change the size of R_r, the dashed line shows the times when we modify the size of O_i, and the dotted line shows the corresponding times for A_{in}. It is important to note that compilation times of the ATL file (ASM compilation) have been omitted because they are constant 120 ms, but these times have been considered to calculate the total execution times of the adaptation process (Figure 21g).

If we focus on the variation in the size of the repository of rules, we can observe that execution time of the transformation in charge of context processing ($\mathcal{C}$) increases proportionally to the number of elements of R_r, but it is below 100 ms for 10,000 elements. Otherwise, the rule transformation process ($\mathcal{R}_\mathcal{T}$), the ATL code extraction ($\mathcal{T}_\mathcal{A}$), the architectural model adaptation ($\mathcal{AMT}_i$), and the validation processes do not depend on the size of R_r and they generate execution times about 30, 40, 10, and 20 ms, respectively. However, time consumed by processes manipulating the repository ($\mathcal{R}_\mathcal{R}$, $\mathcal{R}_\mathcal{S}$ and $\mathcal{R}_\mathcal{L}$) increases proportionally to the number of rules. It may be observed that for repositories with over 4500 rules, the total adaptation processing time surpasses 1000 ms and is up to 2.5 s for repositories with 10,000 elements. Nevertheless, our adaptation rule repository does not have more than 1000 elements because the transformation rules can be generalized to keep a reasonable number of rules. Therefore, execution times required for the adaptation process are within assumable times for our purpose, because 1 s is not an excessive delay for adapting our component-based UIs.

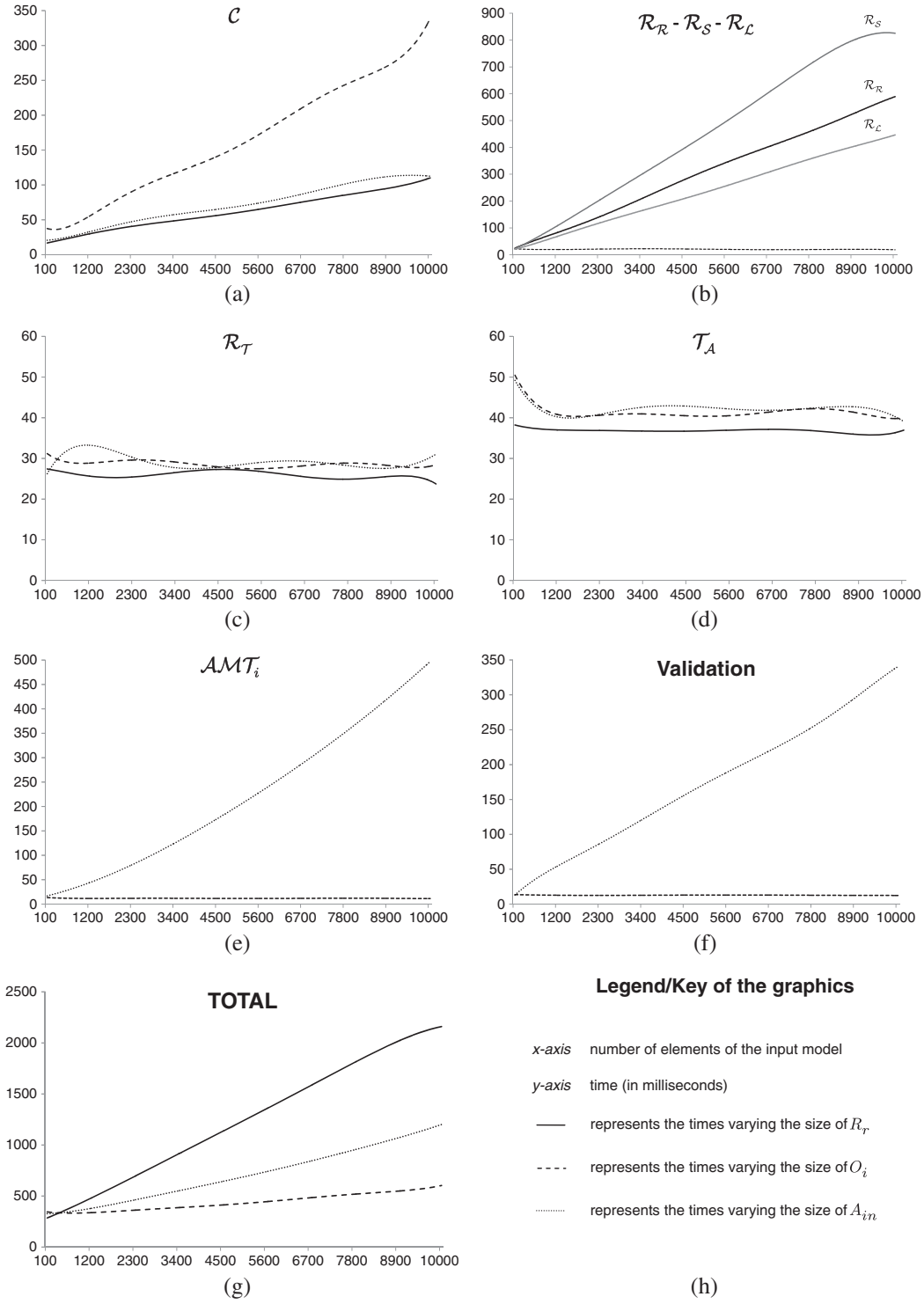


Figure 21. Execution times varying R_r, O_i, and A_{in}.

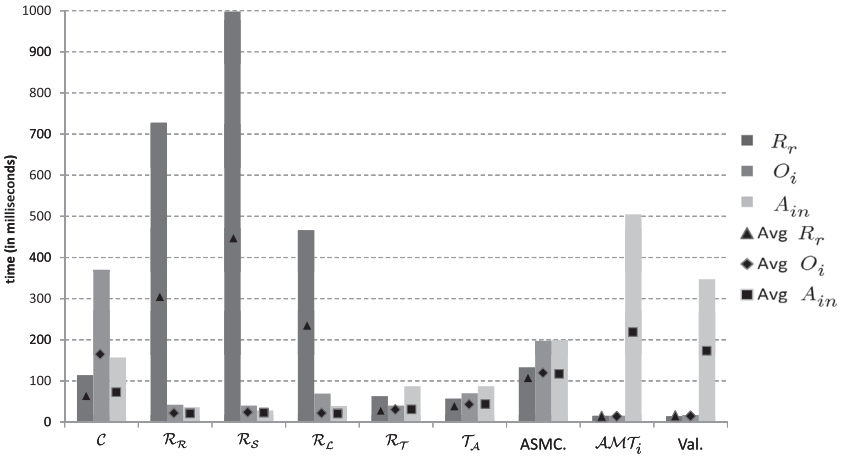


Figure 22. Summary of execution times.

Regarding the execution times varying the number of elements of the O_i ($card(A_{in}) = 100$ and $card(R_r) = 100$), we can observe that the only affected subprocess is the one in charge of processing the context information ($\mathcal{C}$). Therefore, the total execution time depends linearly on the O_i cardinality. Nevertheless, the resulting times are between 300 and 600 ms, what are assumable times considering that our observer models never reach such high sizes. When we vary the size of the A_{in} ($card(O_i) = 10$ and $card(R_r) = 100$), we can observe that three subprocesses of the adaptation schema are affected: processing the context ($\mathcal{C}$), validating the architectural model, and performing the architectural model transformation ($\mathcal{AMT}_i$). The first is linearly affected with a low gradient, whereas the other two processes strongly depend on the A_{in} cardinality, with a gradient close to 1. This causes that the total execution time also depends linearly on the number of elements of the initial architectural model. The resulting times of this experiment are between 300 and 1200 ms, what are assumable times (as in the previous case) taking into account that the size of our architectural models will not exceed 100 elements.

Figure 22 summarizes the execution times generated by each of the subprocesses of the adaptation schema. It represents the maximum times for the aforementioned experiment and the average time for executions from 100 to 10,000 for the size of the input model. In this figure, one can observe the influence of each input model on each subprocess. $\mathcal{C}$ transformation depends on the three input models, but the size of O_i is the factor that has more impact. The size of R_r model is the one that has more influence on the total execution time, as we can see for $\mathcal{R}_{\mathcal{R}}$, $\mathcal{R}_{\mathcal{S}}$, and $\mathcal{R}_{\mathcal{L}}$ subprocesses. On the other hand, $\mathcal{R}_{\mathcal{T}}$, $\mathcal{T}_{\mathcal{A}}$, and ASM compilation subprocesses are not influenced by the size of the models. Finally, the architectural model transformation and the validation subprocesses depend only on the A_{in}, as it is logical. Some other information about the results of the execution times of previous implementations is also available on the web [47].

5.2. Discussion

The adaptation proposal described in this article is based on a series of assumptions. The adaptation target system must be represented in the form of a component-based architecture. These components together describe the system's functionality and are not independent of each other but have dependency relationships in which the behavior of one can affect the rest of the components in the architecture. Furthermore, variations in system context may make it necessary to adapt the architecture, for example, by modifying the connections between components, inserting new ones or deleting existing components, among others. With the idea that adaptation is not static, that is, that the same adapted architecture is always generated by the same changes in context, the adaptation process and its logic must be dynamic and adaptable in order to generate new architectural models that had not been pre-established when the system was designed.

Applying the proposal to a case study, we have chosen GUIs, due to the trend toward Social Semantic Web or Web 3.0, in which UIs make use of technologies to favor exchange of information within the Web community and incorporate capacities that allow users to cooperate and share information. We also intended to achieve smart GUI (SUI), which learn from interaction with the user or group of users, changing and evolving their behavior. Therefore, GUIs are described by means of architectural models containing the specification of UI components, and they can adapt their functionality due to certain changes in the context.

Following these premises, an adaptation schema has been developed based on model transformation, in which both architectures representing system and context information are represented by models. Furthermore, the rules that represent the adaptation operations (e.g., insert component X, delete component Y, connect components X and Y to Z) are also described in models and stored in a rule repository. The sequence of the adaptation schema manipulates these three key pieces by processing the context information, determining the adaptation operations, rating, selecting, and updating the repository rules, and finally constructing an M2M transformation that adapts the starting architectural model at run-time.

All the subprocesses in the adaptation schema were implemented as M2M transformations (using ATL) or TCS extractions, and some generate intermediate elements (such as the subset of selected rules) that are represented in the form of models. These particularities in defining the elements and functioning of our adaptation proposal provide the following *benefits*.

- b#1 System adaptation is dealt with on a high level of abstraction because of the use of models.
- b#2 The models and the OCL restrictions formally define the syntax and some part of the static semantics of the system elements.
- b#3 MDE techniques and tools make model manipulation simple.
- b#4 Proper construction of elements can be checked by applying validation techniques and model checking.
- b#5 The use of model transformation techniques makes it possible to generate well-constructed output models in conformance with their metamodel.
- b#6 Traceability information is generated and it can be used for the maintenance and improvement of the adaptation process.
- b#7 By separating adaptation rules in a repository, new unforeseen adaptation situations can be solved by modifying the repository.
- b#8 Modularization of the different transformation processes in the adaptation schema makes their independent modification possible without having to change the rest of the processes.
- b#9 Run-time construction of the transformation that adapts the architectural models makes possible dynamic adaptation that can evolve.

This last benefit is achieved through the use of HOTs, which allow us to convert any rule selected from the repository into executable ATL code. Therefore, a fixed set of parameterizable adaptation rules is not able to obtain the same flexibility as our adaptation schema. This is because in a fixed set, you cannot define a priori all the possible adaptation rules (even using parameters). It is true that a component-based architecture definition is simple enough to define all the possible rules based on insertion, deletion, connection, and modification operations. However, sometimes, it is not equivalent to execute a rule that includes a number of operations or to execute one rule for each operation. For example, a rule which is in charge of inserting two new components and connecting them it is not exactly the same as three separated and parameterized rules, two for inserting and one for connecting. First, the three separate rules depend on the order, because the components cannot be connected until they are inserted; and second, this separated execution generates worse performance results. Moreover, if the architectural metamodel changes, the parameterized rules and the way to build them must be modified, whereas in our proposal, only the rule repository must be updated (or expanded). In addition, this repository mechanism allows to dynamically add the new rules from the expert (manually) or from the system (automatically).

5.3. Threats to validity

The usage of an MDE approach to the field of adaptation and in particular our proposal definition presents a number of shortcomings or threats to validity which must also be discussed. Even though the adaptation process implements a mechanism for rule conflict resolution, we are not dealing with goal dependencies. Goal conflict resolution requires a new proposal for the representation and the management of the objectives (for example, goal models) and it is not addressed in the proposed adaptation schema. Otherwise, direct manipulation of the models for their modification, management or validation operations makes it necessary to consider the computation load. Furthermore, the MDE model transformation techniques operate on a very abstract level in the definition of system elements, and the tools and libraries currently available to implement these techniques also add a considerable load to computation time.

Regarding the drawbacks highlighted in [57], even we have not represented our possible configurations as states and the reconfiguration operations as transition between states, there is also an explosion in the number of M2M rules that the adaptation process executes. The growing number of rules executed is not directly related to the final transformation adapting the architectural model but affects the processes that are responsible for managing the context information and the rules. This proportional increase depends on the number of context variables and the number of adaptation rules that exist in the rule repository, as shown in Figures 21 and 22. The second flaw of adaptive systems based on MDE is the evolution of the system. In our case, this evolution involves dynamically changing the behavior of our adaptation process. The modular nature of our adaptation schema allows us to address this change independently on each subprocess. However, in the current state of the proposal, it is only possible to modify at run-time the repository of adaptation rules, allowing different behaviors to the same inputs. The logics of the subprocesses (which manage context information) and rules are pre-set and their evolution will be addressed in future work.

Otherwise, even though the syntactic information is represented in the models and some parts or the semantics could be checked through the OCL restrictions, some other semantic information about the architectural models is implicit in the proposed methodology (e.g., in the model transformations) or is addressed in the subsequent process which is responsible for resolving the concrete models from the abstract definition. The first fact could represent a problem for getting a complete overall idea of the whole process, and we attempted to solve it by explaining each step separately. The second fact implies that this paper addressed only the generation and validation of the abstract representation of the architectural models, and the validation of the final component-based system (in the case study, the final UI) is postponed for the later processes.

Another important threat to be taken into account is the management of the composition relationship. In the paper, the example scenario is simple, having few components in the initial and the adapted models. Nevertheless, from our point of view, this simple case study covers all the possibilities that the architectural metamodel offers: simple components, complex components, and connections between components. It is possible to observe that in ATL code generated from the rules of the repository, the management of the containment relationship is accomplished (through the 'component_parentÂt' attribute). Therefore, the management of components' inside components is managed by the transformation rules in the same way, and we understand that it is not necessary to show more containment examples (regardless of the depth level) in order to better understand the process. Regarding the instantiation of the child and parent components, we must remark that our process manages the adaptation at the PIM level of the architectures; for this reason, the implemented mechanism of the child instantiation will be managed at the PSM level. Therefore, our rule repository and our adaptation process are not intended to manage this situation. We reviewed other possible example scenarios, with more components and more containment and connection relationships, but we believe that they will not provide a more illustrative scenario, and in contrast, it will add more complexity to show and explain the adaptation process. Otherwise, it is important to clarify that our proposal is oriented to software architectures built from components with medium or high granularity. Therefore, the final architectures and the final GUIs do not usually contain a large number of components, and the number and the depth level of composition relationships will not be high.

As we explained earlier and along the paper, the proposed adaptation process is applied at the abstract level of the representation of the software architectures (i.e., at the PIM level). Consequently, it is necessary to complete some additional steps in order to reach the final software (the final UI in the domain studied). Therefore, when the abstract architectural model is obtained at the end of the adaptation process, we make two major assumptions: the realization process is able to resolve a valid configuration of concrete components (i.e., a PSM representation) that fulfills the abstract definition, and these concrete components have their corresponding operations available to instantiate them for building the final software. These assumptions can be considered as limitations of the proposal.

Furthermore, our proposal needs to execute some transformation rules in refining mode and therefore it relies on this configuration of ATL. This mode is necessary to avoid the final transformation which handles the adaptation of the model that has to build all the elements from an empty model by modifying or removing only the parts that are changing. This fact implies that the adaptation schema cannot be executed properly without this refining mode.

In addition, for validation and evaluation purposes, it is necessary to carry out some experiments with different sizes of input models and with a large number of these models. Generating all these models by hand is not feasible. For this reason, we programmatically created the required models before launching each experiment. We implemented a process that generates elements with random values between all the possible values that the elements could have. It is not the ideal way to build the test scenarios but it is the only suitable way to construct a wide experiment with a large number of models with a large number of elements with different values.

6. RELATED WORK

As we addressed the adaptation process from a generalist perspective, the works reviewed are related with the adaptation of component-based architectures. Regarding the architectural specifications, the paper developed in [39] shows a survey of self-management dynamic software systems. This work establishes the main elements that a component-based system should have to be considered 'self-managed' or 'self-adaptive': (i) there is a change that starts the change of the system or its adaptation; (ii) a selection process for architectural transformation is performed; (iii) reconfiguration operations are implemented; and (iv) final architecture is evaluated after reconfiguration. Our proposal meets these requirements and it is interesting to compare it with the main aspects highlighted in [39] of other approaches (Table IX). We can see that our proposal meets the main aspects and offers some improvements because it is based on MDE. In addition, our proposal provides greater flexibility because each subprocess is encapsulated in an M2M transformation and it is possible to add new elements to the adaptation schema.

In this regard, it is interesting to compare our proposal with other MDE approaches dealing with dynamic adaptation of software systems to execution context. The rainbow framework [32] provides mechanisms for adapting and updating architectural models to system requirements. Like our proposal, it uses abstract architectural models to monitor, evaluate, and adapt the configurations that are later transferred to the system being run, but these operations are not done using MDE model transformation techniques. This implies that this approach is not benefited from the simplicity of managing the models through the use of a model transformation language. In [58], authors present a proposal for the adaptation of component-based systems where reconfiguration operations and rules are established at design-time using state machines. In our case, reconfiguration operations are selected at run-time from a rule repository. Therefore, this approach cannot easily modify the adaptation logic and this modification has to be done at design-time.

The MADAM/MUSIC approach [59, 60] uses architectural models to describe variability, that is, the models themselves contain the information and criteria for selection so that middleware can derive the adaptation to the context at run-time. This joint representation of the architecture and variability impairs the modification of both elements, thus decreasing its flexibility. In our case, we separate the adaptation logic from the architectural models, and it is represented in model transformations rules in a repository. In [61], variability models are defined to describe the adaptation

Table IX. Comparative with other architectural approaches.

	Formal specification	Change initiation	Types of changes	Selection	Management
Le Métayer	Graph	Internal and external	Basic reconfiguration operations	Constrained from pre-defined set	Centralized
COMMUNITY	Graph	Internal and external	Basic and composite reconfiguration operations	Constrained from pre-defined set	Centralized
CHAM	Graph	Internal and external	Basic and composite reconfiguration operations	Constrained from pre-defined set	Centralized
Dynamic Wright	Process algebra	Internal	Basic and composite reconfiguration operations	Constrained from pre-defined set	Centralized
Darwin	Process algebra	Internal	Add components	Pre-defined	Centralized or distributed
LEDA	Process algebra	Internal	Add components and connectors	Constrained from pre-defined set	Centralized or distributed
PiLar	Process algebra	Internal	Basic and composite reconfiguration operations	Pre-defined	distributed
Gerel	Logic	Internal and external by a tool or infrastructure	Basic and composite reconfiguration operations	Pre-defined Centralized	
Aguirre-Maibaum	Logic	Internal	Basic reconfiguration operations	Pre-defined	Centralized
ZCL	Logic	Internal	Basic reconfiguration operations	Predefined	Centralized
RAPIDE	X/Open language	Internal	Basic reconfiguration operations	Constrained from pre-defined set	Centralized or distributed
Our approach	Model-based	Internal (sensor and triggers) and external (user interaction) stored in observer model	MDE operations allow all types of changes that generate a correct model	Constrained from pre-defined set. Ready to be unconstrained adding and deleting repository rules at run-time.	Centralized, each architecture could represent the subsystem of a particular user within a collaborative system

MDE, model-driven engineering.

logic and thereby separate it from the system functioning. In this case, the adaptation logic is separated from the architectural models but it is neither intended nor defined to be changed at run-time, as our proposal. As a consequence, the adaptation logic is static and cannot be modified at run-time if, for example, new components are incorporated to the system or new adaptation alternatives are discovered.

There are also other approaches dealing with software adaptation that use other MDE techniques, such as graph-based model transformations. In [62], authors describe an approach to develop rule-based refactorings of models. It uses graph-based transformations instead of ATL language and this approach does not manage a rule repository as we do. The authors in [46] present an approach for the adaptation of architectural model based on graph-based model transformations. It describes a metamodel for software systems based on EJB components, so that it addresses adaptation at PSM level. The adaptation is achieved by applying triple graph grammar rules. In contrast, we performed our adaptation at the PIM level of the component-based systems and we have developed a repository of ATL transformation rules from which we built at run-time our model transformation.

In [63], the authors propose implementing the adaptive system control loop as a component-based system that can also be adapted. This sort of control loop can be reconfigured at run-time to incorporate new knowledge dynamically. Our final purpose is to achieve a similar system, in which the logic of adaptation changes according to the knowledge acquired from execution. At this time, our adaptation process can be dynamically adapted by varying the rule repository used. Otherwise, Ramirez *et al.* [64] deal with the important aspect of uncertainty in adaptive systems that can affect requirements, design, and execution stages. In our proposal, as we have constructed the adaptation process in modules and separated the adaptation rules in a repository, we have an updating mechanism available at run-time to improve those aspects that were not detected in earlier stages and without updating affecting other process modules or producing a strong impact on the system.

Another type of adaptive system is dynamic software product line. These systems are similar to the traditional software product lines but variability is linked to run-time [65, 66]. In [24], the authors apply its use to the domain of *smart homes*. In their proposal, they use variability models to activate or deactivate characteristics at run-time, thus complying with context conditions. Therefore, this approach only uses MDE for representation issues, it is not benefited from model transformations and does not provide flexibility mechanism to modify the adaptation operations. In [57, 67], the authors describe a dynamic software product line support architecture controlling the number of variants the system may have. They combine aspect-oriented and model-driven techniques to adapt the models by model-weaving. However, this approach lacks some mechanism to modify the adaptation logic. Instead, we define the variability of our system in the model transformation processes and the repositories that define the adaptation rules. Table X shows the most significant MDE approaches for architectural adaptation that most closely resemble our proposal. Therefore, it summarizes the contributions and differences with our proposal and shows whether they contain the elements that we consider most important in relation to our work.

Some proposals for adaptive systems make use of high-level programming languages for their evolution. Irmert and Fischer [68] and Serral *et al.*c [69] propose implementations based on Java, which is executed in an OSGi platform [70] to adapt the software at run-time. Contrary to our proposal, programs written in programming languages present some lacks related to the structure and type definition, and therefore, programmers have more challenges to write complex adaptation frameworks. However, the use of MDE provides strong typing mechanisms (due to metamodel definitions) which facilitate the implementation of dynamic adaptation solutions.

Concerning the dynamic composition of model transformation, some studies have proposed incremental updating of transformation processes and their dynamic construction from a set of rules [71]. This approach uses *selective linear definite* resolutions to build a transformation tree to transform a source model. It calculates new target models from the modification of the facts associated to the source model and its corresponding modification in the transformation tree. Therefore, this approach does not change the rules within the transformation process as we do.

Table X. Comparative with other MDE approaches.

	Run-time	Rule repository	Rule/goal conflict resolution	Dynamic adaptation logic	Dynamic model transf.	Adaptation mechanism	Main contributions	Differences with our proposal and other remarks
Rainbow [32]	Yes	No	Goals (concerns)	No	No	Adaptation strategies constrained by properties, invariants, and operators implemented in Java.	Adaptation and updating architectural models to system requirements. Abstract architectural models to monitor, evaluate, and adapt the configurations.	No use of MDE model transformation techniques. No benefited from the simplicity of managing models through some transformation language.
AUTOSAR [58]	Yes	No	No	No	No	Design of alternative architectural models and parts of their behavior as state machines.	Reconfiguration operations and rules are established at design-time using state machines.	Reconfiguration operations are fixed and are not variable at run-time. Cannot easily modify the adaptation logic (and it must be done at design-time).
MADAM /MUSIC [59, 60]	Yes	No	No	No	No	Platform independent services for managing applications, components and component instances.	Architectural models to describe variability. The models contain the info and criteria for alternatives and selection.	Adaptation logic is not separated from the architectural models. This joint representation impairs their modification, decreasing the flexibility.
DiVA [61]	Yes	Yes	Rules and Goals	No	No	Aspect-oriented modeling and dynamic weaving techniques.	Variability models to describe the adaptation logic. Property-based policies to manage the constrains and the rules and to select the configuration.	Adaptation logic is separated from the architectural models but is neither intended nor defined to be changed at run-time. This logic cannot be modified if new components are managed or new adaptation alternatives are discovered.
MoRE [24]	Yes	No	No	No	No	OSGi framework to implement reconfiguration operations.	DSPL and variability models to activate or deactivate characteristics at run-time.	Adaptation logic is fixed in variability models. The approach is not benefited from model transformations and does not provide any mechanism to modify the adaptation operations.

Table X. *Continued.*

	Run-time	Rule repository	Rule/goal conflict resolution	Dynamic adaptation logic	Dynamic model transf.	Adaptation mechanism	Main contributions	Differences with our proposal and other remarks
Models@ Run.time [57, 67]	Yes	No	No	No	No	Aspect-Oriented Modeling and dynamic weaving techniques.	DSPL architecture controlling the number of the variants. Aspect-oriented and model-driven techniques are combined to adapt the models. (Described along the paper)	Adaptation logic is fixed in variability models. The proposal lack some mechanism to modify the adaptation logic.
Our approach	Yes	Yes	Rules	Yes	Yes	M2M and HOT transformations using ATL.		

DSPL, dynamic software product line; M2M, model-to-model; HOT, higher-order transformation.

In [49], an approach to transformation composition in ATL is proposed, and it justifies the use of a hybrid transformation language as ATL facilitates the development of adaptive changes because the invocation and definition of the rules are not based only on the explicit call of the rules. In contrast to our proposal, this approach does not represent the rules through models, so that we are able to manage the rules using MDE techniques and we use HOT transformations to generate dynamically the model transformation.

In [72], the author presents a proposal using the QVT transformation language. It describes the mechanisms that this language provides to perform fine-grained and coarse-grained compositions of the transformations. The fine-grained operations allow extracting and reusing the code of the rules existing in the transformations. In contrast, we model the rules in an abstract way and store them in a repository that could be managed by using MDE techniques. In addition, this allows us to incorporate to the rules some attributes and descriptive information that help us in the rule selection process. The article in [50] describes the ATL property to perform rule module superimposition. This mechanism allows to superimpose multiple transformations in a single one, generating as a result a model transformation with the union of all the rules (carrying out overwrite, replacement, and inheritance operations). Our proposal does not apply superimposition mechanisms but the reuse of the rules, and the obtaining of the objective model transformation is based on the selection of rules from a repository to dynamically generate the transformation through the use of a HOT.

Other studies, such as [73, 74], propose updating transformations for refactoring models to adapt them at run-time, but do not try to refactor the transformation itself. One of the purposes of refactoring M2M transformations is to improve or adapt their behavior to the system context by restructuring or adding new rules or *helpers* [36]. Following an MDE approach, such refactoring can be implemented as model transformations in which the transformation itself participates as input and/or output of the transformations, as it occurs in HOT [37, 75]. An interesting proposal for the semi-automatic development of model transformations using HOT transformations is shown in [76]. It describes the MeTAGeM framework, which uses different levels of abstraction to represent the transformations with the goal of building specific M2M processes in different languages at design-time (i.e., as a tool for software engineers). In our case, we make use of HOT to dynamically generate the M2M transformations at run-time for adapting our architectural models.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented, explained, and validated our proposal for run-time adaptation of component-based software architectures. In this proposal, the adaptation of these architectures is not a static process implemented a priori, but it is dynamically constructed from transformation rules selected from a repository. The subset of selected rules can vary according to context information, the current state of the architectures, the adaptation logic implemented to meet the requirements, and the rule repository used.

As the application domain of our proposal, we have selected component-based GUIs that make use of *Social Semantic Web* technologies. This type of UI favors information sharing within the Web community and incorporates intelligence capacities that allow cooperation among users who share common goals. Our definition of the UI can adapt its functionality to the state of the context, obtaining intelligent interfaces or *SmartUIs* that are able to learn from interaction with users and whose behavior evolves. A tool has also been designed to test and validate the adaptation process applied in a possible scenario within the domain of UIs.

The adaptation process has been abstracted by developing a DSL which models the structure of an adaptation schema based on manipulation of models by M2M transformation. Thus, the adaptation schema can be changed in turn (to improve it and update it) by implementing a new model that fits the adaptation metamodel. In our adaptation schema, we have developed a transformation process called $\mathcal{C}$, whose purpose is to determine the optimal adaptation operations from changes in the system context. This is followed by a $\mathcal{R}_{\mathcal{S}}$ process that selects the most appropriate rules according to the adaptation operations needed, based on the information described by the rules in the repository.

To update the information in the rules, we have developed two model transformation processes: (i) one that is in charge of rating the rules ($\mathcal{R}_{\mathcal{R}}$) and (ii) another that updates the values of their attributes depending on the selection process ($\mathcal{R}_{\mathcal{L}}$). Both update the rules at run-time. The difference is that the modifications generated by the former are not permanent but are a mark that is given and disappears when the adaptation process ends. However, the modifications generated by the latter are persistent, thereby making it possible to make use of the information related to the utilization of the repository rules. This information is currently used by the process selecting rules from the repository. In an upcoming version, we will use it to provide our proposal with a decision-making system whose heuristics will use this traceability information. These data, reflected in the adaptation rules along with the context processing logic and the rule selection logic, define the critical points in the adaptability of our proposal. Therefore, the capacity for system adaptation depends on the behavior defined in both logics, in addition to the capacity for updating the rule repository according to events in the system. Finally, once the repository rules have been selected, they are converted into ATL code by applying a HOT-type M2M transformation followed by a TCS extraction. This code constitutes the M2M transformation in charge of adapting the initial architectural model into an adapted architectural model.

We think that our proposal is more than a rule-based approach for dynamic adaptation. Adaptation operations are performed on the abstract level instead of running systems. These operations perform structural modifications on the architecture, for example, inserting a new component, deleting an existing one, and reconnecting some ports of the architecture. Then, at the concrete level, we will develop as future work a trading process that will choose different concrete components from the abstract definition of the architecture, generating the PSM model from the PIM definition, and making possible that it will be able to choose a concrete component or another depending on the system platform.

As previously mentioned, our proposal contains a mechanism for rule conflict resolution. However, it is only in charge of avoiding that the set of selected rules causes a run-time error in the transformation (i.e., there is more than one adaptation rule matching the same element). Therefore, the adaptation goal conflict resolution and the process to check the coherence of the selected rules are to be addressed in future work. Otherwise, our rule selection process makes it possible for the transformation in charge of adapting the architectures to be constructed at run-time based on context and system requirements, that is, not pre-set. In a later stage, we will intend to provide our proposal with a decision-making proposal that will make it more adaptable. Our goal will be to modify the existing transformation rules in the repository or even insert new transformation rules in the system. We are also considering the possibility of improving the description of our components by making use of ontologies. This can assist in the managing of abstract components and concrete component selection (e.g., in the domain of UI components) based on abstract components [77].

In summary, the main contributions of the methodology presented in this paper are the following:

1. The methodology allows us to represent the component-based architectures in an abstract level, both the adaptation rules and the remainder elements of our adaptation schema (context information, intermediate structures, etc.) through models by using MDE.
2. It also provides the definition of all the operations that manage these models through MDE techniques (i.e., through model transformations).
3. It allows us to not have pre-set any final model transformation that is in charge of adapting the architectural model that represents the initial component-based system.
4. It defines a modular adaptation process that obtains this final model transformation.
5. It selects at run-time the appropriate adaptation rules from a rule repository by using the variations of the context information.
6. It updates the repository from the subset of selected rules for next adaptation executions.
7. It makes use of HOTs model transformations to generate at run-time the final model transformation that is in charge of adapting the initial architectural model to the adapted one.

ACKNOWLEDGEMENTS

This work was funded by the EU ERDF and the Spanish Ministry of Economy and Competitiveness (MINECO) under Project TIN2013-41576-R, the Spanish Ministry of Education, Culture, and Sport (MECD) under a FPU grant (AP2010-3259), and the Andalusian Regional Government (Spain) under Project P10-TIC-6114.

REFERENCES

1. Daniel F, Matera M, Yu J, Benatallah B, Saint-Paul R, Casati F. Understanding ui integration: a survey of problems, technologies, and opportunities. *IEEE Internet Computing* 2007; **11**(3):59–66. DOI: 10.1109/MIC.2007.74.
2. Hoyer V, Fischer M. Market overview of enterprise mashup tools. In *Service-Oriented Computing (ICSOC'2008)*, Springer-Verlang, Berlin, Heidelberg, 2008; 708–721. DOI: 10.1007/978-3-540-89652-4_62.
3. Batty M, Hudson-Smith A, Milton R, Crooks A. Map mashups, Web 2.0 and the GIS revolution. *Annals of GIS* 2010; **16**(1):1–13.
4. Beier B, Vaughan MW. The bull's-eye: a framework for web application user interface design guidelines. In *Proceedings of the SIGCHI conference on Human factors in computing systems,* ACM, 2003; 489–496.
5. Galitz WO. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons: New York, NY, USA, 2007.
6. Iribarne L, Padilla N, Criado J, Padilla N, Vicente-Chicote C. Metamodeling the structure and interaction behavior of cooperative component-based user interfaces. *Journal of Universal Computer Science* 2012; **18**(19): 2669–2685.
7. Gruber T. Collective knowledge systems: where the social web meets the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web* 2008; **6**(1):4–13. DOI: 10.1016/j.websem.2007.11.011.
8. Mikroyannidis A. Toward a social semantic web. *Computer* 2007; **40**(11):113–115.
9. O'Reilly T. What is Web 2.0: design patterns and business models for the next generation of software. *Communications and Strategies* 2007; **1**(65):17–37.
10. W3C. W3C Standards: Semantic Web 2010.
11. Iribarne L, Criado J, Padilla N, Asensio J. Using COTS-widgets architectures for describing user interfaces of web-based information systems. *International Journal of Knowledge Society Research* 2011; **2**(3):61–72. DOI: 10.4018/ijksr.2011070106.
12. Iribarne L, Troya JM, Vallecillo A. A trading service for COTS components. *Computer Journal* 2004; **47**(3): 342–357. DOI: 10.1093/comjnl/47.3.342.
13. Cheng BHC, de Lemos R, Geise H, Inverardi P, Magee J. Software Engineering for self-adaptive systems: a research roadmap. In In *Software Engineering for Self-Adaptive Systems*, Vol. 5525, Cheng BHC (ed.). LNCS, Springer-Verlang: Berlin, Heidelberg, 2009; 1–26. DOI: 10.1007/978-3-642-02161-9_1.
14. Salehie M, Tahvildari L. Self-adaptive software: landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 2009; **4**(2):1–42. DOI: 10.1145/1516533.1516538.
15. Sadjadi SM, McKinley PK. ACT an adaptive CORBA template to support unanticipated adaptation. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'2004)*, Tokyo, Japan, 2004; 74–83. DOI: 10.1109/ICDCS.2004.1281570.
16. Blair G, Bencomo N, France RB. Models@run.time. *Computer* 2009; **40**(10):22–27. DOI: 10.1109/MC.2009.326.
17. 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.
18. Bencomo N, Blair G. Using architecture models to support the generation and operation of component-based adaptive systems. In In *Software Engineering for Self-Adaptive Systems*, Vol. 5525, Cheng BHC, de Lemos R, Geise H, Inverardi P, Magee J (eds). LNCS, Springer-Verlang: Berlin, Heidelberg, 2009; 183–200. DOI: 10.1007/978-3-642-02161-9_10.
19. Calvary G, Coutaz J, Thevenin D, Limbourg Q, Bouillon L, Vanderdonckt J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 2003; **15**(3):289–308. DOI: 10.1016/S0953-5438(03) 00010-9.
20. OMG. MDA Guide, Version 1.0.1 2003.
21. Criado J, Vicente-Chicote C, Iribarne L, Padilla N. A model-driven approach to graphical user interface runtime adaptation. In *Proceedings of the 5th International Workshop on Models@run.time*, 2010; 49–59.
22. Iribarne L, Padilla N, Criado J, Asensio J, Ayala R. A model transformation approach for automatic composition of COTS user interfaces in web-based information systems. *Information Systems Management* 2010; **27**(3):207–216. DOI: 10.1080/10580530.2010.493816.
23. Czarnecki K, Helsen S. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003; 1–17.
24. Cetina C, Giner P, Fons J, Pelechano V. Autonomic computing through reuse of variability models at runtime: the case of smart homes. *Computer* 2009; **42**(10):37–43. DOI: 10.1109/MC.2009.309.

25. Seinturier L, Merle P, Rouvoy R, Romero D, Schiavoni V, Stefani JB. A component-based middleware platform for reconfigurable service-oriented architectures. *Software: Practice and Experience (SPE)* 2012; **42**(5):559–583. DOI: 10.1002/spe.1077.
26. Gui N, De Florio V, Holvoet T. Transformer: an adaptation framework supporting contextual adaptation behavior composition. *Software: Practice and Experience (SPE)* 2013; **43**(8):937–967. DOI: 10.1002/spe.2137.
27. Sun J, Zhang YP, Fan J. Towards a context-aware middleware in smart car space. In *Proceedings of he 4th International Conference on Genetic and Evolutionary Computing (ICGEC'2010)*, Shenzhen, China, 2010; 276–279. DOI: 10.1109/ICGEC.2010.75.
28. Fouquet F, Morin B, Fleurey F, Barais O, Plouzeau N, Jezequel JM. A dynamic component model for cyber physical systems. In *Proceedings of the 15th Symposium on Component Based Software Engineering (CBSE'2012)*, ACM, Bertinoro, Italy, 2012; 135–144. DOI: 10.1145/2304736.2304759.
29. Pantsar-Syvaniemi S, Simula K, Ovaska E. *Context-Awareness in Smart Spaces*: Riccione, Italy, 2010.
30. Edwards G, Garcia J, Tajalli H, Popescu D, Medvidovic N, Sukhatme G, Petrus B. Architecture-driven self-adaptation and self-management in robotics systems. In *Proceedings of ICSE 2009 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2009)*, Vancouver, Canada, 2009; 142–151. DOI: 10.1109/SEAMS.2009.5069083.
31. Inglés-Romero JF, Vicente-Chicote C, Morin B, Barais O. Towards the automatic generation of self-adaptive robotics software: an experience report. In *Proceedings of the 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2011)*, IEEE, Paris, France, 2011; 79–86. DOI: 10.1109/WETICE.2011.54.
32. Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* 2004; **37**(10):46–54. DOI: 10.1109/MC.2004.175.
33. Grundy J, Hosking J. Developing adaptable user interfaces for component-based systems. *Interacting with Computers* 2002; **14**(3):175–194. DOI: 10.1016/S0953-5438(01)00049-2.
34. Rodríguez-Gracia D, Criado J, Iribarne L, Padilla N, Vicente-Chicote C. Runtime adaptation of architectural models: an approach for adapting user interfaces. In *Proceedings of the 2nd International Conference on Model and Data Engineering (MEDI'2012)*, Vol. 7602, LNCS, Springer-Verlang, Berlin, Heidelberg, 2012; 16–30. DOI: 10.1007/978-3-642-33609-6_4.
35. Cicchetti A, Di Ruscio D, Eramo R, Pierantonio A. Automating co-evolution in model-driven engineering. In *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC'2008)*, Munich, Germany, 2008; 222–231. DOI: 10.1109/EDOC.2008.44.
36. Wimmer M, Martínez S, Jouault F, Cabot J. A catalogue of refactorings for model-to-model transformations. *Journal of Object Technology* 2012; **11**(2):1–40. DOI: 10.5381/jot.2012.11.2.a2.
37. Tisi M, Jouault F, Fraternali P, Ceri S, Bézivin J. On the use of higher-order model transformations. In *Proceedings of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA'2009)*, Vol. 5562, LNCS, Springer-Verlang, Berlin, Heidelberg, 2009; 18–33. DOI: 10.1007/978-3-642-02674-4_3.
38. Kruchten P, Obbink H, Stafford J. The past, present, and future for software architecture. *IEEE Software* 2006; **23**(2):22–30.
39. Bradbury JS, Cordy JR, Dingel J, Wermelinger M. A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems (WOSS'2004)*, ACM, Newport Beach, CA, USA, 2004; 28–33. DOI: 10.1145/1075405.1075411.
40. Criado J, Iribarne L, Padilla N, Troya J, Vallecillo A. An MDE approach for runtime monitoring and adapting component-based systems: application to WIMP user interface architectures. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'2012)*, Izmir, Turkey, 2012; 150–157. DOI: 10.1109/SEAA.2012.27.
41. Hnětynka P, Plášil F. Using meta-modeling in design and implementation of component-based systems: the SOFA case study. *Software: Practice and Experience (SPE)* 2011; **41**(11):1185–1201. DOI: 10.1002/spe.1036.
42. OMG. Meta-object facility (MOF) specification, v2.4.1 2011.
43. Steinberg D, Budinsky F, Merks E, Paternostro M. *EMF: Eclipse Modeling Framework*. Addison-Wesley: Longman, Amsterdam, 2008.
44. Atkinson C, Stoll D. Orthographic modelling environment. In *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE'2008)*, Springer-Verlang, Berlin, Heidelberg, 2008; 93–96. DOI: 10.1007/978-3-540-78743-3_7.
45. Vogel T, Seibel A, Giese H. The role of models and megamodels at runtime. In *Proceedings of the MoDELS 2010 Workshops*, Vol. 6627, LNCS, Springer-Verlang, Berlin, Heidelberg, 2011; 224–238. DOI: 10.1007/978-3-642-21210-9_22.
46. Vogel T, Giese H. Adaptation and abstract runtime models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2010)*, Cape Town, South Africa, 2010; 39–48. DOI: 10.1145/1808984.1808989.
47. Criado J, Rodrguez-Gracia D, Iribarne L, Padilla N. Towards the adaptation of component-based architectures by model transformation (website). http://acg.ual.es/isoleres/adaptation.
48. Cabot J, Gogolla M. Object constraint language (OCL): a definitive guide. In *Formal Methods for Model-Driven Engineering (SFM'2012)* Bernardo M, Cortellessa V, Pierantonio A (eds)., Vol. 7320, LNCS, Springer-Verlang, Berlin, Heidelberg, 2012; 58–90. DOI: 10.1007/978-3-642-30982-3_3.

49. Kurtev I, van den Berg K, Jouault F. Rule-based modularization in model transformation languages illustrated with ATL. *Science Computer Programming* 2007; **68**(3):138–154. DOI: 10.1016/j.scico.2007.05.006.
50. Wagelaar D, Van Der Straeten R, Deridder D. Module superimposition: a composition technique for rule-based model transformation languages. *Software and Systems Modeling* 2010; **9**(3):285–309. DOI: 10.1007/s10270-009-0134-3.
51. Warmer JB, Kleppe AG. *The Object Constraint Language: Getting Your Models Ready For MDA*. Addison-Wesley: Boston, MA, USA, 2003.
52. Jouault F, Allilaire F, Bézivin J, Kurtev I. ATL. A model transformation tool. *Science of Computer Programming* 2008; **72**(1):31–39. DOI: 10.1016/j.scico.2007.08.002.
53. Insfran E, Gonzalez-Huerta J, Abraho S. Design guidelines for the development of quality-driven model transformations. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'2010)*, Springer-Verlang, Berlin, Heidelberg, 2010; 288–302. DOI: 10.1007/978-3-642-16129-2_21.
54. Jouault F, Bézivin J, Kurtev I. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings 5th International Conference on Generative Programming and Component Engineering (GPCE'2006)*, ACM, Portland, Oregon, USA, 2006; 249–254. DOI: 10.1145/1173706.1173744.
55. Arora S, Hazan E, Kale S. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing* 2012; **8**(1):121–164.
56. Jouault F, Kurtev I. Transforming Models with ATL. In *Proceedings of model transformations in practice workshop at MoDELS'2005*, Springer-Verlang, Berlin, Heidelberg, 2006; 128–138. DOI: 10.1007/11663430_14.
57. Morin B, Barais O, Jézéquel JM, Fleurey F, Solberg A. Models@run.time to support dynamic adaptation. *Computer* 2009; **42**(10):44–51. DOI: 10.1109/MC.2009.327.
58. Becker B, Giese H, Neumann S, Schenck M, Treffer A. Model-based extension of AUTOSAR for architectural online reconfiguration. In *Proceedings of the 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems*, Springer-Verlang, Berlin, Heidelberg, 2009; 83–97. DOI: 10.1007/978-3-642 -12261-3_9.
59. Floch J, *et al.* Playing MUSIC – building context-aware and self-adaptive mobile applications. *Software: Practice and Experience (SPE)* 2013; **43**(3):359–388. DOI: 10.1002/spe.2116.
60. Rouvoy R, Eliassen F, Floch J, Halssteinsen S, Stav E. Composing components and services using a planning-based adaptation middleware. In *Software Composition* Pautasso C, Tanter (eds)., Vol. 4954, LNCS, Springer, Heidelberg, 2008; 52–67. DOI: 10.1007/978-3-540-78789-1_4.
61. Fleurey F, Solberg A. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'2009)*, Springer-Verlang, Berlin, Heidelberg, 2009; 606–621. DOI: 10.1007/978-3-642-04425-0_47.
62. Becker B, Lambers L, Dyck J, Birth S, Giese H. Iterative development of consistency preserving rule-based refactorings. In *Proceedings of the 4th International Conference on Theory and Practice of Model Transformations* Cabot J, Visser E (eds)., Vol. 6707, LNCS, Springer, 2011; 123–137. DOI: 10.1007/978-3-642-21732-6_9.
63. Perrouin G, Morin B, Chauvel F, Fleurey F, Klein J, Le Traon Y, Barais O, Jézéquel JM. Towards flexible evolution of dynamically adaptive systems. In *Proceedings of the 34th IEEE International Conference on Software Engineering (ICSE'2012)*, Zurich, Switzerland, 2012; 1353–1356. DOI: 10.1109/ICSE.2012.6227081.
64. Ramirez AJ, Jensen AC, Cheng BHC. A taxonomy of uncertainty for dynamically adaptive systems. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2012)*, Zurich, Switzerland, 2012; 99–108. DOI: 10.1109/SEAMS.2012.6224396.
65. Hallsteinsen S, Hinchey M, Park S, Schmid K. Dynamic software product lines. *Computer* 2008; **41**(4):93–95. DOI: 10.1109/MC.2008.123.
66. Hinchey M, Park S, Schmid K. Building dynamic software product lines. *Computer* 2012; **45**(10):22–26. DOI: 10.1109/MC.2012.332.
67. Morin B, Barais O, Nain G, Jézéquel JM. Taming dynamically adaptive systems using models and aspects. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'2009)*, Vancouver, Canada, 2009; 122–132. DOI: 10.1109/ICSE.2009.5070514.
68. Irmert F, Fischer T. Meyer-Wegener K. runtime adaptation in a service-oriented component model. In *Proceedings of the ICSE 2008 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2008)*, ACM, Leipzig, Germany, 2008; 97–104. DOI: 10.1145/1370018.1370036.
69. Serral E, Valderas P, Pelechano V. Supporting runtime system evolution to adapt to user behaviour. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'2010)*, Vol. 6051, LNCS, Springer-Verlang, Berlin, Heidelberg, 2010; 378–392. DOI: 10.1007/978-3-642-13094-6_30.
70. OSGi Alliance. OSGi service platform, rel. 4.1 2007.
71. Hearnden D, Lawley M, Raymond K. Incremental model transformation for the evolution of model-driven systems. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'2006)*, Springer-Verlang, Berlin, Heidelberg, 2006; 321–335. DOI: 10.1007/ 11880240_23.
72. Belaunde M. Transformation composition in QVT. In *Proceedings of the First European Workshop Composition of Model Transformations (CMT'2006)*, Bilbao, Spain, 2006; 39–45.
73. Kolovos DS, Paige RF, Polack F, Rose LM. Update transformations in the small with the epsilon wizard language. *Journal of Object Technology* 2007; **6**(9):53–69.

74. Porres I. Rule-based update transformations and their application to model refactorings. *Software and Systems Modeling* 2005; **4**(4):368–385. DOI: 10.1007/s10270-005-0088-z.
75. Tisi M, Cabot J, Jouault F. Improving higher-order transformations support in ATL. In *Proceedings of the 3rd International Conference on Model Transformation (ICMT'2010)*, Vol. 6142, LNCS, Springer-Verlang, Berlin, Heidelberg, 2010; 215–229. DOI: 10.1007/978-3-642-13688-7_15.
76. Bollati VA, Vara JM, Jiménez A, Marcos E. Applying {MDE} to the (semi-)automatic development of model transformations. *Information and Software Technology* 2013; **55**(4):699–718. DOI: 10.1016/j.infsof.2012.11.004.
77. Paulheim H, Probst F. Ontology-enhanced user interfaces: a survey. *International Journal on Semantic Web and Information Systems* 2010; **6**(2):36–59. DOI: 10.4018/jswis.2010040103.