Department of Informatics
University of Almería

A Trading and Model-Based Methodology for Adapting Dynamic User Interfaces

THESIS

Presented by

Javier Criado Rodríguez

for the degree of
PhD in Computer Science

Thesis supervised by

Dr. Luis Iribarne
Full University Professor
Department of Informatics
University of Almería

Dr. Nicolás Padilla
Full University Professor
Department of Informatics
University of Almería

Almería, July 13th, 2015

Written and edited by: Javier Criado
Printed by: Murex Factoría de Color (Almería, Spain)

July 2015

Summary

We are currently witnessing constant evolution in all areas of computer technology, especially the software development. This has led to applications that need to adapt to both the new technology and the new emerging trends present in each area of the Computer Science community. The software applications most seriously affected by this need to evolve are those which provide a *User Interface* (UI) and (to an even greater extent) the *Graphical User Interfaces* (GUI). This is because these interfaces provide the main link between users and software and, as such, they should be optimized and adapted to the people who use them. In this sense, it is sometimes necessary to focus on just improving specific areas when adapting the user interface; for example, to show new information, hide unnecessary data, or change what is being shown or how it is shown.

User interfaces are often one of the most updated elements during the lifetime of an application and this usually means that they are the area where most time is spent during the software life cycle. Therefore, new techniques and methodologies are being created to facilitate the adaptation of such interfaces. A specific type of user interface is a Web-based User Interface. Such interfaces are often affected because the information which currently exists on the Internet is highly globalised. An example of this assertion (and the motivation for the following case of user interface adaptation) is when a web user interface accesses resources from external applications to the system. This access must be properly managed to avoid execution or visualization errors. A possible adaptation operation of the user interface would involve checking that all web resources are available and updating those which are not accessible (modifying the access addresses or swapping resources for alternatives).

The origin of the motivation for the adaptation of web user interfaces may be different from the right access management to external resources. Let us suppose that all resources are accessible but a new resource, which improves on an existing user interface resource, has turned up in a different location. In this case, it may be interesting to substitute one element for another one and even cease to use the original resource (in the whole application) while employing the new one. Another scenario that gives rise to adaptation may be determined by the context conditions of the user interface. Supposing that the main memory available by the system is insufficient to support some of the functionalities of the interface, for example, because it requires more memory than it is available. In such a situation, it would be necessary to disable or inhibit this functionality so that the user interface can keep on working, even though its full functionality might be reduced.

The aforementioned situations are examples in which the adaptation of user interfaces is aimed at ensuring their proper implementation. However, there is another relevant

approach that is important to highlight, which is the adaptation of the "interface" to the user preferences and the user behaviour. The information obtained from the interaction with the interface can be used as a source of information for the adaptation. Let us suppose that one user adds another resource every time they use a specific resource present in the user interface (assuming that such a mechanism exists) because this new resource is needed and is complementary to the first one. If this operation is repeated often, it becomes clear that the automatic inclusion of the second resource would be an ideal application of the adaptation mechanism. For this reason, it is interesting to incorporate an adaptive mechanism that allows for changes or upgrades to be made to the interface inspired by the events produced by users.

In many cases, the examples of adaptation mentioned are not practical or useful if made at design time, because the changes are motivated by actions occurring at run-time. Therefore, the research presented here focuses on a dynamic adaptation of the user interfaces. The application domains are consequently limited since modifications at run-time need to be allowed by the user interfaces without affecting or interrupting their working. Therefore, the user interfaces that are pre-compiled or interpreted, whose parts can be reconfigured to adapt dynamically, are chosen. The domain of web user interfaces previously mentioned is an ideal field for the application of this research.

A particular type of web user interfaces are *mashup* interfaces [Daniel et al., 2007]. These interfaces are built from the composition of different parts or pieces, creating a single interface with which the user interacts. Such parts do not relate to text fields or buttons, but identify components of a medium or high (*i.e.*, coarse) granularity and encapsulate some type of functionality. Other studies have proposed a UI built from parts. Nevertheless, the components of these interfaces behave isolatedly and are not dependent on each other and consequently, the actions and interactions run on a component have no impact on the rest. In this regard, it may be beneficial to incorporate any type of relationship between components of an interface in order to carry out, for example, the aforementioned adaptation case in which a component always works together with another component. Thus, the user interface would be represented as an architecture describing which components are present and the relationships between them.

The developed work focuses on dynamically adapting *mashup* user interfaces taking into account both the information from the execution environment and the knowledge of the domain. From this information, it is worth highlighting the user profiles and the user preferences. This research aims to get intelligent user interfaces or *smart GUI* which are able to reconfigure their component architecture at *run-time* and *self-adapt* to different scenarios. To do this, the construction of the user interface and reconfiguration operations will be conducted from third-party component repositories, such as COTS (*Commercial Off-The-Shelf*) components. These can be modified over time, for example, by removing existing or adding new components to repositories.

Furthermore, we believe that the user interfaces, in spite of being the main objective of our research, are not the only domain which can benefit from the examples of adaptation mentioned. An application domain could be considered to be any software system built from components which needs to be adapted at run-time by reconfiguring its architecture, and in which relationships between its components exist. For this reason, the research undertaken maintains a "generalist" approach aimed at the dynamic adap-

tation of component architectures. The focus on intelligent user interfaces (*i.e.*, smart GUI) can be extended to other different domains related to smart devices, such as green buildings, home automation, smart TVs, smart cities, robotics, etc.

The generalist approach requires the descriptions of the component architectures to be applicable to any domain. This can be solved through abstraction mechanisms such as representation by modeling techniques. By applying an engineering approach based on models, component architectures could be represented by a PIM (*Platform Independent Model*) and the existence of adaptation operations at this level and on these types of models is possible. On the other hand, when using a PIM it would be necessary to determine which specific components best comply with the component definition included in the architecture, this way obtaining a PSM (*Platform Specific Model*). This framework identifies the three main activities of our methodology: (a) the definition of user interfaces from component architectures, (b) the adaptation of the component architectures at a level of representation which is independent from the platform, and (c) the configuration of component architectures at a level of representation which is dependent on the platform.

The first activity, *i.e.*, the definition of a user interface, must include the descriptions of the components of which it is composed (at a functional, extra-functional level, etc.) in addition to the relationships between the components of the architecture. Thus, there are languages which define user interfaces from their components. However, these languages lack notations with enough information to be able to carry out adaptation operations using the components' characteristics or their inter-dependencies.

Regarding the second activity, there are some mechanisms to adapt PIM architectures, for example, through the application of model transformation techniques. However, the execution of model transformations at run-time is an emerging field of research and there are no proposals for the run-time transformation of component architectures. Furthermore, in current transformation methods, the logic of the transformations taking place is static and, *a priori*, is pre-determined. Nevertheless, the adaptation required must allow the dynamic modification of the adaptation rules in order to be able to dynamically define new change strategies and adapt to new situations at run-time.

Regarding the third activity, there are mediation services that facilitate or automate component selection and search processes. There are also models of mediation for the development of software systems using the construction of architectural configurations. However, the existing procedures for developing component-based software lack the mechanisms to allow the building of software at run-time. In this regard, neither the component documentation models nor the procedures for computing configurations are aimed at dynamically constructing or reconfiguring architectures.

Objectives and contribution

The development of a process for adapting dynamic user interfaces is influenced by the following general objectives of this research:

1. Firstly, it is essential to develop a methodology and experimental framework aimed at adapting the user interfaces at run-time. Each of the processes involved must be

identified (*i.e.*, all the required functionality must arise), as well as the elements operating as input and output of such processes.

2. Secondly, it is necessary to make a proposal for describing user interfaces using component architectures. This representation must be carried out by using modelling techniques and also be extendible to other domains.

3. Thirdly, we must develop an automatic process which generates a new definition of adapted architecture from an initial architecture and some context information. This transformation process must allow the dynamic variation of adaptive logic and must be run in an acceptable time for the adaptation of the architecture to be done at run-time.

4. Fourthly, it is necessary to develop an automated process that, from all possible component configurations, generates the best configuration to comply with an input architecture definition. This process must also be run in an acceptable time, to achieve the user interface adaptation dynamically.

5. Finally, an evaluation of the proposed methodology must be performed by applying it to a sample scenario and validating the results and times obtained.

The main contribution of the developed research work is a proposed methodology for adapting mashup-type user interfaces, which are described, or can be described, from component-based architectures. This adaptation is done at *run-time* and its behaviour is not immutable, but rather, is capable of carrying out different adaptation operations when faced with similar input situations. This dynamic characteristic is achieved mainly by using *model transformation* mechanisms and *mediation services* (*i.e.*, component trading techniques). The two main disciplines of the Software Engineering involved in this research are Model-Driven Engineering (MDE) and Component-Based Software Development (CBSD). The concrete contributions of this work are listed below:

— A methodology based on *Model-Driven Engineering* (MDE) techniques and mediation techniques (also known as *trading* services in Software Engineering literature) has been defined for the adaptation of user interfaces at run-time. This methodology consists of two stages: a first stage of *transformation* in which the platform independent component architectures (which we call *abstract architectures*) are adapted, and a second phase of *regeneration* in which the platform dependent architectures (which we have called *concrete architectures*) are built from the definitions of abstract architectures.

— A *Domain Specific Language* (DSL) has been built for the definition of the main elements involved in the methodology, that is, the abstract architecture element, the concrete architecture, the abstract components and the concrete components. This language has been developed by means of meta-models and the use of EMF (*Eclipse Modelling Framework*), and all the processes of the methodology manipulate instances of the language. By using these modelling techniques, restrictions for defining the models have also been established and validated using the OCL language (*Object Constraint Language*).

— The transformation process (the first stage of the methodology) has been developed as a set of transformations designed to dynamically generate a new *model-to-model* transformation (M2M) every time the abstract architectures are adapted. All the changes have been implemented using ATL (*ATL Transformation Language*), and the dynamic transformation is built at run-time from a repository of rules and HOT (*Higher-Order Transformation*).

— A DSL language has been built to annotate the M2M transformation rules. This language allows to define a catalogue of properties. Although the language can be applied to any transformation language, a concrete syntax has been proposed to annotate the ATL language rules. Furthermore, a tool for creating catalogues and annotations has also been developed. The function of this language is to improve the annotation process of the repository rules used in the transformation stage of the methodology.

— The regeneration process (second stage of the methodology) has been developed as a component mediation service, which extends the traditional ODP (*Open Distributed Processing*) trading model for objects and services. This mediation service allows the realization of single/complex component searches, as well as the addition, modification and deletion of component specifications from component repositories associated to the mediation service. It also provides a mechanism to configure their execution policies. This functionality corresponds to the implementation of the *Lookup*, *Register* and *Admin* interfaces of the ODP mediation standard. Additionally, this service has extended the traditional mediation model incorporating a new interface *Configs* to resolve concrete architectures from definitions of abstract architectures at run-time. Two algorithms have been implemented to resolve the configurations: one recursive and another one, based on a type A* search algorithm, which improves the performance of the first algorithm.

— Finally, for the evaluation and validation of the methodology, different test scenarios have been developed to adapt the user interfaces of a GIS (*Geographic Information System*). In addition, repositories have been built for storing example architectures and components used in such scenarios. The implementation of the two stages of the methodology has been encapsulated into web services so that it can be invoked at any time thus facilitating its debugging and execution. Meanwhile, the evaluation of both main processes has been conducted by measuring their performance and using some existing and developed tools.

Background and future lines

The development of this thesis is part of two research projects, one funded nationally by the Ministry of Economy and Competitiveness (MINECO) with reference TIN2010-15588, and another one funded regionally by The Andalusian Government with reference ID P10-TIC6114. The first project is titled "*i-SOLERES: A methodology for the retrieval and exploitation of environmental information through evolutive and cooperative user*

interfaces". The main objective of this project, as its name suggests, was the exploitation of GIS information systems. For this, a key element was the development of user interfaces that were capable of changing both over time and depending on the users interacting with the system. In this way, the current research project helped to complete the research related to the objective and the sample scenarios were also used as case studies in the doctoral thesis.

The second project is titled "*ENIA: Development of an intelligent Web agent of environmental information*". The objective of this project is strongly linked to the development of a trading service capable of managing both environmental information and the components that are part of the UI system. This trading service is encapsulated within the Web agent who incorporates a certain intelligent functionality such as the anticipation of user actions or the evolution of the logic responsible for adapting user interfaces. In this sense, the work presented here has played an important role in the development of mediation services which manage interface components and perform reconfiguration operations. Furthermore, like the project mentioned above, this project has also served as a sample domain for the testing and development of case studies.

The research presented has been developed under a University Lecturer Training (Formación del Profesorado Universitario) grant with reference AP2010-3259 and funded by the Ministry of Education, Culture and Sports (MECD, Spain). During this grant, research work related to the above projects has been developed as well as other tasks which have led to the development of this thesis. This thesis has been developed within the Computer Science Doctoral Program Ref-8705 of the RD1393/07 (with Mention of Excellence) from the University of Almeria, Spain.

From the objectives listed in the previous section, a number of lines of future investigation are available, such as the application of the methodology to different domains; not only dynamic UI, but also home automation environments, robotics, etc. Another possible line is the extension of the UI types that are used. Our intention is to use another type of UI along with Web based user interfaces, such as *Natural User Interaction* interfaces (NUI). In addition, our dynamic transformation proposal draws from a repository of rules which is intended to be modified over time, for example by incorporating new rules or modifying and eliminating existing rules.

This can be achieved by studying the interaction of the user and the other events collected by the system in order to infer new rules to be incorporated into the repository. Some of these lines form part of other research work being conducted by the research group to which the author of this work belongs. This will lead to other doctoral theses. Other lines of investigation have been included in a new research project funded by the MINECO with reference TIN2013-41576-R. This project is titled "*Evolving dynamic systems in the cloud: A framework toward the smart user interfaces*" and is currently in its early stages of development.

Conclusions

The constant evolution of software development technology involves adapting existing applications with regards to both the emerging techniques and new trends. The user interfaces of applications are affected by this need for evolution since they are the main link between software and users and, on occasion, are the part of the software most affected by the latest updates, since they are the "visible part of the applications.

This need for evolution gathers even more relevance in web user interfaces since the external resources which they access may change their location or not work correctly. It is also possible that new resources appear which enhance the functionality of existing resources, or whose context requires the incorporation or disabling of some element in the user interface. To this end, the component-based web user interfaces (also called *mashup* user interfaces), are an ideal application domain for the development of mechanisms which permit adaptation. These types of user interfaces can be adapted by reconfiguring the component architecture. This reconfiguration can be done at run-time since in the Web environment it is possible to make use of dynamically interpreted user interfaces.

The research developed in this thesis has presented a methodology for adapting mashup user interfaces at run-time. The methodology establishes the execution of two stages: a transformation process based on model transformation operations (which adapt the component architectures), and a regeneration process based on a mediation service which allows the calculation of the component configurations which best meet a particular architecture. In this chapter, we summarize the results obtained from this work. The chapter is divided into four sections. Firstly, the contributions that the research work provides to the scientific community are described. Subsequently, the limitations of the proposed methodology are identified. Then, the research lines that remain open are presented. Finally, a list of publications resulting from the research work is provided.

Main contributions of the approach

The main contribution of the research is a methodology developed to adapt mashup user interfaces at run-time. These user interfaces are described based on component architectures at different levels of representation. The two levels of representation involved in the methodology are the abstract level and the concrete level. A component and, by extension, an architecture of components at the abstract level, describes the set of features desired to be present in the user interface. On the concrete level, the components (and

architectures) described have an equivalent in the real world and have been selected to be part of the user interface.

The adaptation accomplished by the application of the methodology is based on two main stages: (1) a process for transforming those abstract architectures which describe an initial user interface, and (2) a process for regenerating concrete architectures that best meet abstract definitions obtained from the previous transformation. This adaptation is done in run-time and the result obtained is not always the same; it depends on several factors. Dynamic adaptation is achieved by introducing variability into the processes of transformation and regeneration used in the methodology.

The transformation process builds a model transformation from a repository of transformation rules. The rules selected depend on the current state of the user interface, context information and the transformation rules available. Regarding the process of regeneration, the concrete architectures which are built depend on: the abstract architecture used as a reference, the concrete components available at any given time, and the heuristics used for the selection and evaluation of the component configurations. The specific contributions derived from this research are as follows:

— A methodology based on Model-Driven Engineering (MDE) techniques and mediation (*trading*) techniques for the adaptation of user interfaces at run-time has been defined. This methodology has two stages: a first stage of transformation in which the adaptation of platform independent component architectures takes place (which we call abstract architecture), and a second phase of regeneration in which the platform dependent architectures are built (which we have called concrete architectures) from abstract architecture definitions [Iribarne et al., 2010a] [Criado et al., 2010a] [Iribarne et al., 2010b] [Iribarne et al., 2011] [Iribarne et al., 2012] (*Chapter 2*).

— A *Domain Specific Language* (DSL) has been built to define the main elements involved in the methodology: abstract architectures, concrete architectures, abstract components and concrete components. This language has been developed through the use of metamodels and EMF (*Eclipse Modeling Framework*). All the processes of the methodology manipulate instances of this language. Through the use of these modeling techniques and OCL (*Object Constraint Language*), restrictions have been set which allow the proper definition and validation of the models (*Chapter 2*).

— The transformation process (the first stage of the methodology) has been developed as a set of transformations designed to dynamically generate a new model-to-model transformation (M2M) every time the abstract architectures are adapted. All the transformations have been implemented using ATL (*ATL Transformation Language*) and the dynamic transformation is built at run-time from a repository of rules and a HOT (*Higher-Order Transformation*) transformation operation [Rodríguez-Gracia et al., 2012] [Criado et al., 2014] (*Chapter 3*).

— A DSL has been built to annotate the M2M transformation rules. This language permits the definition of a catalogue of properties. Although the language can be applied to any transformation language, a concrete syntax has been proposed to annotate the ATL language rules. Furthermore, a tool for creating catalogues and

annotations has also been developed. The function of this language is to improve the annotation process of the repository rules used in the transformation stage of the methodology [Criado et al., 2015b] (*Chapter 3*).

— The regeneration process (second stage of the methodology) has been developed as a mediation service which extends the traditional trading services. This mediation service allows component searches from input parameters and the addition, modification and deletion of specific components. It also provides a mechanism to configure their execution policies. This functionality corresponds to the implementation of the *Lookup*, *Register* and *Admin* interfaces of the ODP (*Open Distributed Processing*) mediation standard []. Additionally, by incorporating a new interface *Configs*, this service extends the traditional mediation model in order to resolve concrete architectures from definitions of abstract architectures in run-time. Two algorithms have been implemented for the resolution of configurations: one recursive and one based on a type A* search algorithm which improves the performance of the first algorithm [Criado et al., 2013a] [Criado et al., 2015a] (*Chapter 4*).

— A mechanism for transforming specific architectures without having to carry out the two main stages of the methodology (*i.e.*, transformation and regeneration processes) has been proposed. This M2M transformation can be used to perform concrete adaptation operations on the user interfaces: changing the value of a component attribute, replacing a concrete component with another, or deleting a concrete component of the architecture [Criado et al., 2011] [Criado et al., 2012] (*Chapter 4*).

— Finally, for the evaluation and validation of the methodology, different test scenarios have been developed to adapt the user interfaces of a GIS (*Geographic Information System*). In addition, repositories have been built for storing example architectures and components used in the scenarios. The implementation of the two stages of the methodology has been encapsulated into web services so that they can be invoked at any time, thus facilitating their debugging and execution. Furthermore, the evaluation of both main processes has been conducted by measuring their performance. Tools for validating the processes implemented have also been developed and used. The results obtained on the maximum execution time (around 1200 milliseconds for both processes), suppose a maximum execution time of 2'4 seconds. These maximum time values are within acceptable limits for the dynamic adaptation of user interfaces [Criado et al., 2013b][Criado et al., 2014] [Vallecillos et al., 2014] [Criado et al., 2015a] (*Chapter 5*).

Limitations of the methodology

Although the approach provides numerous contributions, there are also some threats to validity which should be discussed. There are a number of limitations associated with the proposed methodology which have implications both in the transformation process of the abstract architectures, and the regeneration process of the concrete architectures:

— Although the representation mechanisms for components and architectures at an abstract level is associated with the level of the *Platform Independent Models* (PIM), in the definition of abstract components it is possible to set some attributes that bring the description closer to that of a *Platform Specific Model* (PSM). This is because the abstract level of the methodology is used to define types of components and, therefore, allows the component types to establish restrictions or conditions close to the final implementation, such as: the type of platform, the type of repository, or the author of the component.

— A major restriction is the management of the composition relationship. Due to the characteristics of the components and the type of mashup user interfaces utilized in the proposed methodology, all the components which form part of the architectures are found at the same level. This does not mean that there are no interface components that are composed of other components, but rather that such elements are managed as a black box in which the reference component is the father. As a result, architecture reconfiguration operations do not include composition operations at run-time. However, by using the relationships between components, a solution is able to be offered which can establish composition relationships using the dependencies between the interfaces and the component ports.

— With respect to component instantiation of the architectures, it should be noted that the methodology assumes a mechanism exists to obtain real objects from a component repository. In our example scenario, *widgets* from W3C and a Wookie repository have been used containing such elements for building mashup user interfaces. The real components are created from the URI described in the concrete components specifications. However, no solutions are offered to create other types of real user interface components or for the instantiation of components in other domains.

— Experiments carried out to evaluate and validate the methodology need to make use of models which have different sizes and are formed by a higher number of elements. The manual generation of these models is not feasible, so they have been automatically created programmatically before the execution of each experiment. To do this, a process has been implemented that generates elements with random values taken from all the possibilities. As a solution for creating models this is not ideal, but it is a proper way to build experiments large enough with a high number of models and containing a large number of elements with different values.

— The models used for defining the architectures (that describe the initial user interfaces) are constructed manually using the reflexive editors obtained from the metamodels of the proposed languages. However, it would be desirable to have a graphical editor which allows the definition of these architectures, thus facilitating the design and development of the user interfaces involved in the methodology. The same applies to the construction of the specifications of abstract and concrete components.

Focusing on the transformation process of the methodology, applying engineering modeling techniques to the field of adaptation and, in particular, the adaptation scheme proposed in this process, has some limitations which should be considered:

— Although the transformation process implements a mechanism for conflict resolution during the selection of rules, the possible conflict that may exist with adaptation objective dependencies is not addressed. The resolution of objective conflicts requires a new approach in order to represent and manage those goals, such as goal models.

— The manipulation of the models in order to carry out creation, modification, management and validation operations, implies a high computational load that must be taken into consideration.

— Transformation process operations operate at a fairly high level of abstraction, as regards the definition of the user interfaces involved. This can mean a considerable difference with respect to the real user interface which is obtained at the end of the application of the methodology and can hinder the compression of the process.

— The current tools and libraries implementing model transformation techniques are still in intermediate stages of development, and are not yet in commercial use. This means that the computation times obtained are relatively high.

— With regard to the problems highlighted in [Morin et al., 2009a]; although the methodology does not represent the configurations as states or the configuration operations as transitions between states, there is an explosion in the number of M2M rules that the adaptation process must run depending on the circumstances. The growing number of rules is not directly related to the dynamic transformation which adapts the abstract architectures, but it does affect the operations responsible for managing the rules and processing the context information.

— Another limitation of basing adaptive systems on MDE techniques is the evolution of the system. In the proposed methodology, this evolution implies the behavior of the transformation process changing dynamically with the modular nature of the proposed scheme allowing each sub process to be changed independently. However, in the current state of the methodology, it is only possible to change (at run-time) the repository of transformation rules.

— The transformation process needs to execute some rules in refining mode of ATL, and is therefore dependent on this type of ATL configuration. This mode is necessary to prevent the transformation from having to build all the elements from an empty model. This implies that the transformation process of the methodology cannot function properly without this mode of execution.

Regarding the regeneration process of the methodology, a number of areas for improvement exist which it is necessary to discuss, with the aim of establishing the limits of the developed trading service:

— The regeneration process of the methodology makes two assumptions that must be taken into account. Firstly, it is assumed that the regeneration process will always find a valid component configuration (at least in the functional part) which complies with the abstract definition. Secondly, in order to build the end user interfaces, it is assumed that the concrete components described by the existing specifications in the repository have corresponding operations of instantiation.

— The development and implementation of the *Lookup* interface of the trading service is a simplification of the definition of the interface which provides the ODP mediation function. Instead of using a template based on the establishment of the service type (*ServiceTypeName*), restrictions (*Constraint*), policy sequence (*PolicySeq*), etc., a message in SQL format is used for the query. This also implies that, to be able to consult the concrete component repository, the processes or entities which make use of the query operation must understand the SQL syntax. To run properly, the SQL query must also focus on obtaining a set of specifications for individual components, rather than other elements of the chosen database.

— The operations developed when implementing the *Register* interface of the trading service include the insertion (*export*), change (*modify*) and elimination (*withdraw*) of component specifications. However, there are no operations to delete using constraints (*withdraw_using_constraint*), describe a component offer (*describe*), or obtain other *Register* interfaces from trading objects with remote locations (*resolve*).

— The *Admin* interface of the developed trading service only allows the establishment of certain management policies, such as the maximum cardinality of the queries, the time limit in the calculation of the configurations, the value of the factors acting in the calculation heuristics, or the minimum ratio values for the calculation of the compliance with the architecture. However, no operations are included for the management of other trader policies to comply with the ODP mediating function, such as the default search cardinality, the matching cardinality, or the cardinality of the elements returned in a query.

— The mechanism established for the semantic matching of the interfaces starts with the condition that there are a number of predefined types, at least for each of the kinds used in the abstract component managed by the system. In this sense, the developed methodology does not offer a mechanism for the consultation and management of the available types.

— In addition to the predefined interface types, it is also necessary to include predefined component types, *i.e.*, abstract components, in the repository. If not, it is not possible to carry out the definition of abstract architecture or use these definitions to construct concrete architectures.

— Experiments with A* configuration algorithms show a very low performance (*i.e.*, execution times are too long) when there are a large number of possible candidate components. Therefore, on these occasions it is beneficial to change the heuristic function calculation without considering the distance value between the start node and the end node ($g(x)$). This converts the heuristic search method into a *greedy* algorithm instead of an algorithm considered to be strictly A*. This means that the execution of the algorithm no longer ensures that the solution produced is the closest solution to the initial node. However, this problem can be limited by controlling the number of components added to the possible solutions and by carrying out an evaluation of valid solutions at a functional level. This means the given solution is always a valid solution although it will not always be the optimal solution.

This limitation must be assumed since the execution times required for the dynamic adaptation occur without causing an excessive delay in the user interface.

— In cases where the concrete component repository size is large (more than one thousand components), a query to obtain the set of component candidates for a concrete architecture takes longer than one second to be processed. This can reach approximately 9 seconds (for repositories with ten thousand components and abstract architectures with ten components). As such, it is necessary to use memory storage (*caching*) mechanisms to obtain candidate components from the database.

Despite the number of limitations which have been detected and mentioned, it is important to highlight that the purpose of this analysis is to limit the scope of validity of the developed methodology and isolate areas for improvement to be addressed in future research work.

Open research lines

From the research work and some of the limitations described, a number of areas of research that are open and can be addressed as future research work have been identified:

— Develop a dynamic composition mechanism which allows the creation (at run-time) of complex user interface components from simple mashup components. In addition, the new complex components created should be uploaded to the repository of concrete component specifications as well as the combination of simple components of which they are constituted. Additionally, these elements should also be uploaded to the widget repository so they are available in other adaptation situations.

— Carry out the implementation a graphical tool which permits the construction and editing of component specifications (abstract and concrete), and component architectures (also at an abstract and concrete level). It would also be interesting if this tool were able to link architectures with the initial interfaces of different users and user profiles that participate in the system.

— Improve the mechanisms which enable the evolution of the developed methodology, especially with respect to the transformation of abstract architectures. In the current state, the logic of the operations that handle the processing of context and rule management (score, select, update, etc.), are pre-set. As future work, the logic of the operations mentioned is intended to also be defined by rules that are obtained dynamically from a rule repository. One rule repository will exist for the context processing operation, another for scoring rules, and another for the rule selection process, etc. This way, it is possible to achieve a methodology which itself can be dynamically adapted.

— Obtain the inference of new rules for the logic which perform the transformation of abstract architectures, for example, from user interaction with the system. Thus, the transformation process is intended to be able to learn, adding new adaptation rules at run-time and acquiring “intelligent” behavior.

— Implement a conflict resolution mechanism for the adaptation purposes, for example using goal models. Thus, if, during the transformation process, a set of adaptation operations is obtained which would make no sense to be executed simultaneously (for example, insert and delete the same component), it will be possible to determine a set of consistent adaptation operations.

— Improve implementation of the three interfaces of the proposed mediation service (*Lookup*, *Register* and *Admin*), in order to comply strictly with the ODP trading function. For example, improving the query operation of the *Lookup* interface or adding configuration attributes which can be managed from the *Admin* interface.

— Extend the trading service by incorporating the functionality relative to the *Link* and *Proxy* interfaces of the trading function of the ODP specification. Thus, the stand-alone mediation service proposed could be considered a full-service type trader.

— Develop a functionality which forms part of the methodology and also allows the management of both of the component types which intervene in the user interfaces (*i.e.*, abstract components), such as the types of data used as input and output in the operations of the functional interfaces of the components.

— Improve the descriptions of the components, the component types and the types of data used for their functional interfaces using ontologies. This may be an advantage when looking for similar elements (for example, using synonyms) or to improve the calculation of component matching.

— Reduce the execution time obtained in the transformation and regeneration processes. On the one hand, parallelization techniques can be applied, and the libraries used in the implementation of both processes can be kept updated (given that new versions may reduce execution times). On the other hand, there is the possibility of developing new versions of the proposed algorithms.

Currently, some of these lines of research are being addressed as part of other doctoral thesis which are being developed within the author's research group and by the supervisors of the present document. A set of the future work mentioned has been incorporated as objectives in the national research project TIN2013-41576-R: "Evolving dynamic systems in the cloud: A framework toward the smart user interfaces".

Publications derived from the thesis

This section presents the articles, book chapters and contributions to conference proceedings which have been published as a result of the research carried as part of the development of this thesis. The following list contains the set of publications in chronological order:

— Iribarne, L., Padilla, N., **Criado, J.**, Asensio, J. A., and Ayala, R. (2010). A Model Transformation Approach for Automatic Composition of COTS User Interfaces in

Web-based Information Systems. *Information Systems Management*, 27(3):207–216. Taylor & Francis. DOI: 10.1080/10580530.2010.493816.

— **Criado, J.**, Padilla, N., Iribarne, L., and Asensio, J. A. (2010). User Interface Composition with COTS-UI and trading approaches: Application for Web-based Environmental Information Systems. *Communications in Computer and Information Science*, Vol. 111, pp. 259–266. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-16318-0_29.

— Iribarne, L., Padilla, N., **Criado, J.**, and Vicente-Chicote, C. (2010). An Interaction Meta-model for Cooperative Component-Based User Interfaces. In *Proceedings of the Second International Workshop on Information Systems in Distributed Environment (ISDE/OTM2010)*, LNCS 6428, pp. 259–268. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-16961-8_44.

— **Criado, J.**, Vicente Chicote, C., Iribarne, L., and Padilla, N. (2010). A Model-Driven Approach to Graphical User Interface Runtime Adaptation. In *Models@Run.Time*, CEUR-WS Vol 641, pp. 49–59.

— Iribarne, L., **Criado, J.**, Padilla, N., and Asensio, J. A. (2011). Using COTS-Widgets Architectures for Describing User Interfaces of Web-Based Information Systems. *International Journal of Knowledge Society Research (IJKSR)*, 2(3):61–72. IGI Global. DOI: 10.4018/ijksr.2011070106.

— **Criado, J.**, Iribarne, L., Padilla, N., Troya, J., and Vallecillo, A. (2011). Adapting Component-based User Interfaces at Runtime using Observers. In *Proceedings of the 16th Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2011)*, pp. 707–712.

— Rodríguez-Gracia, D., **Criado, J.**, Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2011). Adaptive Transformation Pattern for Architectural Models. In *Proceedings of the 16th Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2011)*, pp. 727–740.

— Iribarne, L., Padilla, N., **Criado, J.**, and Vicente-Chicote, C. (2012). Metamodeling the Structure and Interaction Behavior of Cooperative Component-based User Interfaces. *Journal of Universal Computer Science (J.UCS)*, 18(19):2669–2685. DOI: 10.3217/jucs-018-19-2669.

— **Criado, J.**, Iribarne, L., Padilla, N., Troya, J., and Vallecillo, A. (2012). An MDE Approach for Runtime Monitoring and Adapting Component-Based Systems: Application to WIMP User Interface Architectures. In *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'2012)*, pp. 150–157. IEEE. DOI: 10.1109/SEAA.2012.27.

— Rodríguez-Gracia, D., **Criado, J.**, Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2012). Runtime Adaptation of Architectural Models: An Approach for Adapting User Interfaces. In Proceedings of the Second International Conference on Model

and Data Engineering (MEDI'2012), LNCS 7602, pp. 16–30. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-33609-6_4.

— **Criado, J.**, Iribarne, L., and Padilla, N. (2013). Resolving Platform Specific Models at Runtime Using an MDE-Based Trading Approach. In *Proceedings of the Fifth International Workshop on Information Systems in Distributed Environment (ISDE/OTM2013)*, LNCS 8186, pp. 274-–283. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-41033-8_36.

— **Criado, J.**, Rodríguez-Gracia, D., Iribarne, L., and Padilla, N. (2013). AMADATL: A tool for dynamically composing new model transformations at runtime. In *Proceedings of the 18th Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2013)*, pp. 367–370.

— Vallecillos, J., **Criado, J.**, Iribarne, L., and Padilla, N. (2014). Dynamic Mashup Interfaces for Information Systems Using Widgets-as-a-Service. In *Proceedings of the Sixth International Workshop on Information Systems in Distributed Environment (ISDE/OTM2014)*, LNCS 8842, pp. 438–447. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-662-45550-0_44.

— **Criado, J.**, Rodríguez-Gracia, D., Iribarne, L., and Padilla, N. (2014). Toward the adaptation of component-based architectures by modeltransformation: behind smart user interfaces. *Software: Practice and Experience*. John Wiley & Sons, Ltd. DOI: 10.1002/spe.2306.

— **Criado, J.**, Martínez, S., Iribarne, L., and Cabot, J. (2015, *In Press*). Enabling the reuse of stored model transformations through annotations. In Proceedings of the 8th International Conference on Model Transformation (ICMT'2015), LNCS 9152. Springer-Verlag Berlin Heidelberg.

— Rodríguez-Gracia, D., **Criado, J.**, Iribarne, L., and Padilla, N. (2015, *In Press*). A collaborative testbed web tool for learning model transformation in software engineering education. *Computers in Human Behavior*. Elsevier. DOI: 10.1016/j.chb.2014-.11.096.

— **Criado, J.**, Iribarne, L., and Padilla, N. (2015, *In Press*). Mediación semántica A* basada en MDE para la generación de arquitecturas en tiempo de ejecución. In *Proceedings of the 20th Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2015)*.