IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 24, 2021, accepted January 4, 2022, date of publication January 11, 2022, date of current version January 14, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3141702*

# Transformation Architecture for Multi-Layered WebApp Source Code Generation

**RICARDO TESORIERO[1], ALEJANDRO RUEDA[1], JOSE A. GALLUD[1], MARIA D. LOZANO[1], AND ANIL FERNANDO[2], (Senior Member, IEEE)**

[1]Albacete Research Institute of Informatics, University of Castilla-La Mancha, 02071 Albacete, Spain
[2]Department of Computer and Information Sciences, University of Strathclyde, Glasgow G1 1XQ, U.K.

Corresponding author: Ricardo Tesoriero (ricardo.tesoriero@uclm.es)

This work was supported in part by the Ministry of Science, Innovation and Universities, Spain, through the national project under Reference RTI2018-099942-B-I00; in part by the Junta de Comunidades de Castilla-La Mancha (JCCM) Regional Government through the Project TecnoCRA under Reference SBPLY/17/180501/000495; and in part by the European Regional Development Funds (FEDER).

**ABSTRACT** The evolution of Web technologies leads to software premature obsolescence requiring technology-independent representations to increase the reuse rates during the development process. They also require integration into service-oriented architectures to exchange information with different Web systems supporting runtime interoperability. Web Applications (WebApps) run on devices with different capabilities and limitations increasing the complexity of the development process. To address these challenges, different proposals have emerged to facilitate the development of WebApps, which is still an open research field with many challenges to address. This paper presents a model transformation architecture based on software standards to automatically generate full stack multi-layered WebApps covering Persistence, Service, and Presentation layers. This transformation architecture also generates the set of test cases to test WebApp business logic. The proposed transformation architecture only requires a UML platform-independent class model as an input to generate fully functional Web applications in a three-tier architecture including the three layers, while most proposals focus on the generation of the Presentation layer. In addition, this architecture employs software industry standards to enable an easy integration into third-party tools and development environments. The transformation Architecture proposed has been empirically validated on the case study of a fully functional travel management WebApp that is generated using a UML class diagram employing a third-party tool integrated into the same integrated development environment.

**INDEX TERMS** Software product lines, computer-aided software engineering, client-server systems.

## I. INTRODUCTION

The development of modern Web systems is becoming increasingly complex. Some critical factors to consider are: (a) the support of runtime interoperability with third-party systems as part of Service Oriented Architectures (SOA) employing Web service APIs; (b) the premature WebApp obsolescence leading to the re-writing of the same application functionality using different software technologies; and (c) the heterogeneity of Web technologies to develop these applications. For instance, in the development of multi-layered WebApps, the persistence layer can be implemented using a relational or a non-SQL database management system, the Web service API to access the system functionality can be implemented following a Representational State Transfer (ReST) API architecture [1] using JSON, or SOAP specifications [2] using XML, and the WebApp User Interface (IU) should take advantage or mitigate the limitation of device features such as device display resolution.

To overcome these challenges, and contribute to the need of providing a framework to generate complete WebApps covering the three development layers, this proposal presents a transformation architecture based on the OMG [3] Model-driven Architecture (MDA) [4] standard to enable stakeholders, analysts, designers, and developers to automatically generate multi-layered WebApps source code using model transformations.

This approach deals with: (a) the runtime interoperability deriving multi-layered WebApp source code defining the Web service layer to access WebApp functionality; (b) the

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

technology obsolescence decoupling the WebApp Platform Independent Model (PIM) defined by a UML class model [5], [6] from the source code defining the Platform Specific Model (PSMs) generated by the model transformation architecture; and (c) the heterogeneity of Web technologies employing different model transformations according to the layer target technology. As a consequence of adopting this approach, from the WebApp technological perspective, the same WebApp UML class diagram can be reused as an input of different transformation architectures to generate different Persistence, Test, Web service and Presentation layers deployment configurations.

For instance, the Persistence layer could employ a relational database management system such as MySQL [7], PostgreSQL [8], SqlServer [9], etc. using the SQL query language to access database information; or could employ a non-relational management systems such as the MongoDB [10] using the JSON format pattern to access database information. The Test layer could be implemented using different Test frameworks such as PHPUnit [11] or Mocha [12].

The Web service layer providing access to the Persistence layer from the Presentation layer and third-party applications in SOA environments could employ a ReST architecture [1] or a SOAP specification [2] implementation. And the Presentation layer representing the WebApp UI could employ a server-side rendering approach using JavaServer Pages (JSP) [13], PHP [14] or JavaServer Faces (JSF) [15]; or a client-side rendering approach using AngularJS [16] or Vue.js [17]. Even using the same rendering strategy, different implementation alternatives could be used; for instance, to render client-side Web pages the Material [18] or Polymer [19] UI frameworks can be used.

The consequence from the software development perspective is the easy integration with third party tools following the OMG MDA standards such as the Eclipse Integrated Development Environment (IDE) [20] which provides developers with a set of software technologies supporting model definition, and both, model-to-model (M2M) and model-to-text (M2T) transformation definition and execution.

To fulfil these requirements, this article presents a set of Eclipse IDE plugins enabling analysts, designers, and developers to generate multi-layered Web systems source code (i.e. Presentation, Web service, Test, and Persistence layers) from UML class models employing a Model-driven Development (MDD) approach using a transformation architecture following OMG MDA standards.

The rest of the paper is structured in the following sections. Section II presents existing approaches in this research field and highlights the research gaps that our proposal aims to address. Section III describes the process to generate WebApps using the proposed transformation architecture. The set of M2M and M2T transformation rules are explained in Section IV. The validation of the proposal as a result of applying these transformations in the case study of a Travel Management WebApp is presented in Section V. Section VI presents a discussion where this proposal is compared to the most relevant related works in the research area. Finally, Section VII presents the conclusions and future works.

## II. RELATED WORKS

This section presents existing approaches in this research field, with the purpose of identifying the main shortcomings and the research gap and provide a better understanding on how the proposed solution addresses the shortcomings. Later, in section VI we present a table with a comparison of the main features of each proposal to highlight the main differences with respect to the approach presented in this paper.

In WSGUI [21] and Dynvoker [22] employ Web service annotations in WSDL [23] or WADL [24] to derive initial versions of UIs that are manually refined to obtain fully functional WebApps. The annotation concept was improved in [25], where authors propose an integrated modelling approach for the task-driven development of interactive service-oriented applications which introduces annotations as part of the Web service description covering the semi-automatic generation of task models and UIs following the OMG MDA standard. However, this approach does not cover the generation of WebApp persistence or business logic layers, or any tool support.

The importance of using models to improve WebApp development process in analyzed in [26] where authors present the preliminary study results of a prototype architecture created with the purpose of using a domain-driven approach to shorten the development of software projects. The modeling of conceptual, navigational, and user interface features of WebApps can be derived from UML profiles as exposed in [27]. The use of UML Profiles to derive WebApps following a model-driven development approach is employed in [28], where authors propose a UML profile for modeling WebApps at the Platform Specific Model (PSM) level to derive Servlets, JSP pages, and Java source code templates. The proposal presented in [29] defines a component-centric UML profile for modeling WebApps that use ASP, JSP, PHP, Servlets, and JavaBeans technologies. Other proposals, such as [30], employ UML Profiles for Web 2.0 mashups using map and Web feed services; and [31] employ them for the generation of Google App Engine applications. However, none of these proposals generate the persistence layer nor provide supporting tools following OMG MDA standards.

The proposal presented in [32] presents a UML profile for AngularJS to build models of AngularJS WebApps, and a set of transformations that transform the model into a code template following OMG MDA standards. However, this approach does not cover the generation of WebApp persistence layer; instead it generates mock-ups. The generation of WebAPIs to provide runtime interoperability is also another missing feature of this approach.

Other approaches employ different types of models, such as the authors of [33] that fuse Workflow, Web 2.0, SOA and WS-BPEL to create a distributed computing environment (ACtive E-commerce Framework called ACEF) to create an inter-operable infrastructure that leverage the migration code

to support multi-platform web service mobile applications. However, it lacks model interoperability, or tool support because it does not follow any standard to provide software development integration.

An interesting proposal where authors employ a cloud platform spreadsheet to create WebApps is [34]. However, it generates monolithic applications limiting the WebApp technology customization versatility (i.e. persistence, web service, and presentation technologies).

The authors of [35] define an MDA to generate UIs for the Android platform based on a UML class diagram. In [36], authors present a proposal for high-quality code generation for low-cost mobile phones. A proposal that envisages the creation of Rich Internet Applications (RIAs) that are visually appealing and ''responsive'' is presented in [37]. From the point of view of the MDD process, authors of [38] present a MDA-based development process to generate the source code of applications for the Android platform.

An interesting DSL proposal for the development of Web applications that defines a Computation Independent Model (CIM), a Platform Independent Model (PIM), and a Platform Specific Model (PSM) is presented in the [39]. However, this proposal is a work in progress and it currently presents no evidence of generating the source code for multi-layer Web applications.

A proposal based on the definition of a set of transformations and DSLs that generate the source code of Web applications for language learning using a MDA approach is presented in [40] and [41].

Some proposals employ automatic source code generation to develop advanced UI configurations, such as distributable user interfaces [42], is presented in [43]. An example of using MDD to generate this type of UIs in the education domain is presented in [44]. However, these proposals are limited to the generation of the presentation layer source code only. They do not generate any source code for the data access or Web services layers. Unlike this proposal, the lack of a comprehensive solution does not enable the automatic generation of the association between layers.

A proposal that uses MDA to generate source code using a Model-View View-Model (MVVM) architecture for Windows Phone application development is presented in [45].

Another proposal that addresses a similar problem to the one proposed in this article is presented in [46]. This proposal presents a transformation model to generate a PSM from a PIM based on a UML class diagram. It generates a single XML file containing the application specification for the PHP CodeIgniter framework [47].

A proposal that combines class and sequence UML diagrams for an application to build MVC Web applications is presented in [48].

While these proposals generate source code for the data access and presentation layers, they do not generate Web service layer source code. The main disadvantage of existing proposals compared to the one presented in this article is the lack of interoperability at both design-time and run-time.

The lack of design-time interoperability is mainly due to the failure to follow OMG standards to enable third-party tool integration and application communication. For example, the proposal presented in this article uses Eclipse Papyrus IDE to generate UML class models that are used by model transformations to generate the application source code.

On the other hand, the lack of interoperability at run-time is due to the absence of a Web service layer to access the information stored in the application database through a standard external interface (e.g. ReST using JSON) limiting the ability to interact with other applications in a service oriented architecture (SOA) scenario. A direct consequence of this lack of interoperability is the inability to deploy 'responsive' Web applications with rendering in the Web client.

One of the advantages of the solution proposed in comparison with using DSLs or dedicated UML profiles is that we use the well-known OMG standard UML class diagrams as a DSL for the proposed transformation architecture input parameter, lowering the learning curve of developers preventing them from learning a new DSL to model the system to be generated. However, our approach lacks the semantic expressiveness that might be provided by DSLs or dedicated UML profiles which would provide a higher level of customisation of the generated code (e.g. customized labels, input constraints, etc.).

To sum up, the main limitations found in existing proposals are that some of them does not cover the three layers, others does not comply with international standards, and others lack of multi-layer integration. Other of the main limitations of existing proposals compared to the one presented in this article is the lack of interoperability at both design-time and run-time. The proposal we present addresses these limitations in an integrated way, generating the three layers (presentation, web service and persistence, and an additional test case layer) as we show in the next sections.

## III. OVERVIEW OF THE TRANSFORMATION ARCHITECTURE

The development process of the transformation architecture defines a set of M2M and M2T transformations to generate the source code for each layer of the WebApp to be developed (i.e. Presentation, Web services, and Persistence layers as well as a set of Test cases to check Persistence layer).

While the M2T transformations have been implemented in ACCELEO [49], the M2M transformations have been developed in ATL [50], [51].

Figure 1 depicts a simplified overview of the model transformation architecture proposed to develop multi-layered WebApps using a single UML class model as the transformation input to generate Persistence, Web service, Presentation, and Test case source code. This development process is based on five model transformations which are depicted in the diagram using blue rectangles. Note that the generation of the WebApp source code requires the generation of an intermediate model in TagML to generate the system UI.

While four of these transformations are M2T transformations that generate the WebApp source code for different

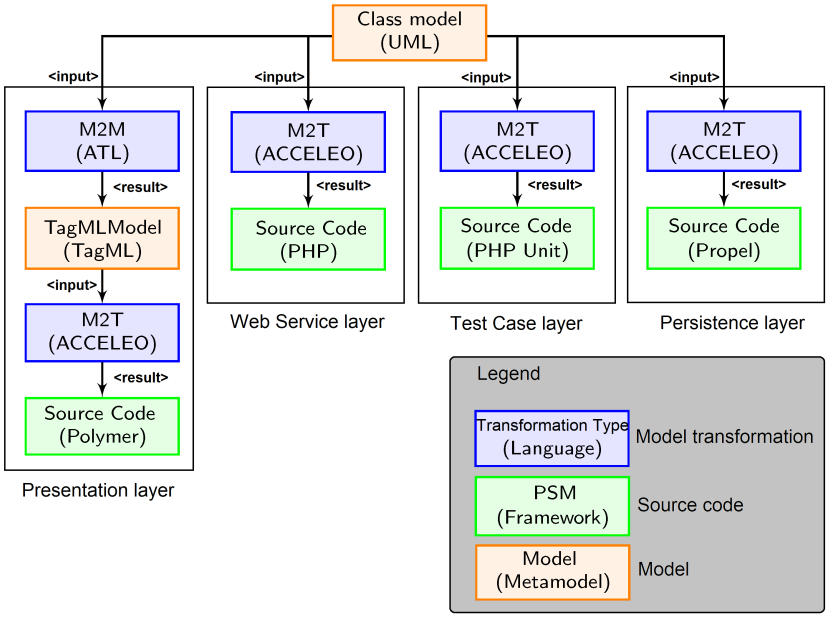


**FIGURE 1. A simplified overview of the model transformation architecture to develop multi-layered WebApps.**

layers, the M2M transformation is defined to generate the presentation model in TagML [52]. Besides, three of the four M2T transformations and the M2M transformation employ the UML class PIM of the WebApp to be developed as input parameter. This UML class model is generated using the Eclipse Papyrus [53] plugin for the Eclipse IDE which proves how this proposal can be easily integrated with third-party tools.

Therefore, this model can also be created or manipulated using a third-party tool with the following OMG MDA standards: UML [54], XML Model Interchange (XMI) format [55], and Meta-Object Facility (MOF) [56]. Although strictly speaking the remaining M2T transformation does not use the UML class model as an input model directly, it indirectly uses it because it is a model generated by the M2M transformation that takes UML class model as an input parameter.

From a technological perspective, while the generated WebApp employs MySQL as the relational database manager for the system and PHP [14] as scripting language on the server side of the application, HTML and JavaScript are used on the client side. In addition, the generated WebApps use XML to render the configuration files.

### A. PERSISTENCE LAYER

The persistence layer code is generated by two M2T transformations that take a UML class model as an input. While the first transformation generates the code responsible for manipulating the information contained in the database, the second transformation generates the test cases code to check the code generated by the first transformation.

The generated code that manipulates the database information employs the Propel object to relational database persistence framework [57] to generate the MySQL database schema and the PHP classes required to manipulate this information in an XML file (i.e., `schema.xml`).

The test cases code employs the PHPUnit framework [11], which is responsible for testing the create, read, delete, and update (CRUD) records functions for the database, as well as the management of the associations between objects implemented by the PHP classes generated by the first transformation that employs the Propel framework to manage table relationships. All test cases are generated in a single PHP script file containing a test case generated for each PHP class.

### B. WEB SERVICE LAYER

The remaining M2T transformation that directly takes the UML class model as an input parameter generates the Web service layer code to access the system database through the persistence layer. The generated Web services follow a Representational State Transfer (ReST) [1] service architecture implemented in PHP that uses the CRUD functions of the persistence layer.

The result of this transformation is a PHP file for each class that relates the HTTP [58] request types defined in the ReST API to the set of operations for each class in the persistence layer. The set of generated classes, in addition to implement CRUD functions, implement the skeleton of the operations defined in the UML class model, as well as all the functions responsible for implementing associations between classes; for example, adding or removing an item from a multi-valued property, and assigning an instance of a class to a property and reflecting it in the database through the persistence layer.

In order to relate class operations to the persistence layer function, the `script.php` file defines a set of high-level functions called from the generated ReST API methods.

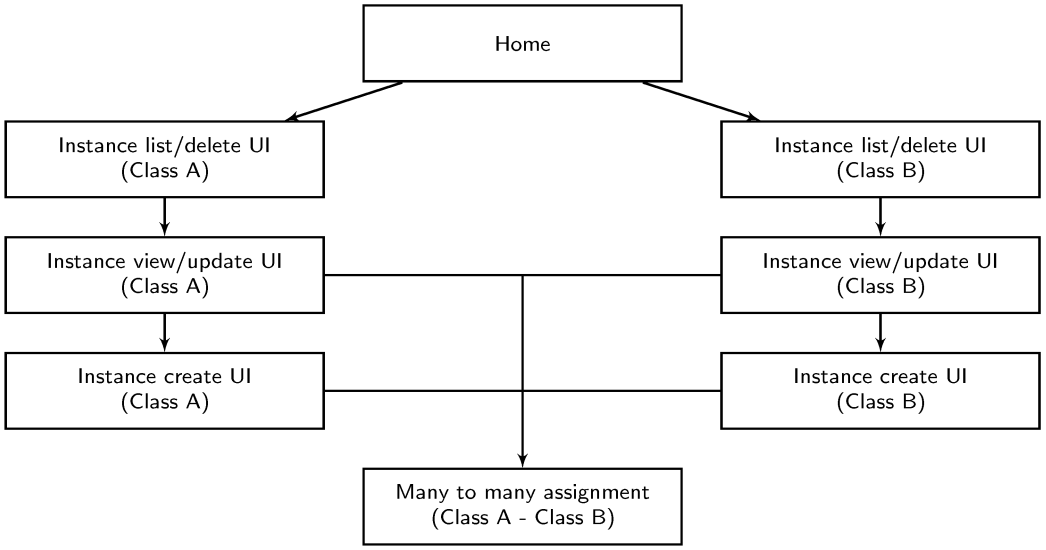


**FIGURE 2.** Meta navigation of the Web system UI including the CRUD management of class instances, one-to-one, one-to-many and many-to-many relationships.

Therefore, this file functions play the role of Mediator [59] between HTTP requests and persistence layer functions.

### C. PRESENTATION LAYER

The M2M transformation that generates the presentation layer code (i.e., WebApp UI) renders the data accessed form the Web services layer generated by the M2T transformation mentioned in the previous section. The result of this transformation generates a TagML model that is passed as input parameter to the M2T transformation generating the presentation layer source code that defines the Implementation Specific Model (ISM) of the system (i.e., a PSM that is specific for a particular representation).

The generated code uses the version 2.0 of the Polymer framework [19] to render the application UI. This framework implements a Web component-oriented language in XML to render the client UI.

The application UI defines a Web component as the Web system main menu (or home page) where users access instances of any class in the UML model. In addition, for each class in the UML model, a Web component is generated to display the list of instances stored in the database. Each item in the list displays the primary key for each instance in order to provide a standard identification (i.e., the name property value of the class instance it represents).

Users access a Web component to view, or modify, class instance attributes containing primitive type values stored into the database.

The UI to manipulate one-to-many associations presents the list of class instances owned by association owner. Thus, users access these instances by clicking on the item in the list that represents it.

The manipulation of many-to-many associations requires the generation of a different Web components where cross-instance associations are set through two lists showing the list of available instances that can be associated, and the list of those instances that are already associated. The implementation of this Web component relies on the functionality of the *data-binding* feature provided by the Polymer framework to manage table relationships in relational databases.

The last UI generated Web component displays a Web form to add new instances to the database. This component is accessed when users click on "Add" button in the Web component that displays the list of instances.

An overview of the navigation pattern through all Web components of the system is depicted on the meta-navigation diagram in Figure 2.

Finally, the information required by Web components is accessed through Web service layers. The connection between the Web services and requests from the UI is carried out automatically, there is no need to modify the generated source code (except for customization purposes).

### D. DEPLOYMENT

Due to the complexity of the development process, transformations automatically generate a set of scripts to relate and configure all layers of generated Web applications. These configuration scripts are generated for the Bower package manager [60] which is used to deploy the generated Web system.

## IV. TRANSFORMATION DEFINITIONS

This section introduces the method based on the architecture presented in Section III to generate WebApp source code.

### A. PERSISTENCE LAYER TRANSFORMATION RULES

The Persistence M2T transformation defines a set of transformation rules to transform UML class model elements (i.e., classes, properties, and associations) into relational database schema elements (i.e. tables, attributes, and

relationships). The following subsections describe how different types of elements are transformed.

#### 1) UML CLASS TO DATABASE TABLES

The following code in ACCELEO M2T transformation language shows a fragment of the transformation rule that transforms UML classes into database tables in the XML file used by Propel to generate the database schema (i.e., `schema.xml`).

```
[template public generateElement(e:Class)]
<table name="[e.name/]"
  phpName="[e.name.toUpperFirst()/]" [if
  (e.isAbstract)] abstract="true" [/if]
  allowPkInsert="true">
```

**LISTING 1. Transformation rule generating tables.**

This rule assigns the class name to the `name` and `phpName` attributes to set the MySQL table name and the PHP class name that cope with table operations. It also sets the `abstract` attribute to `true` if the UML class is defined as abstract.

#### 2) CLASS PROPERTY TO TABLE ATTRIBUTE

The following code in ACCELEO shows a fragment of the transformation rule that turns properties in UML classes into table attributes in the XML script used by Propel to generate the database schema file. It derives the `column` tag `size` attribute from UML class property name. It also derives the column type from class property type using the `e.generateTipo()` rule which turns UML primitive types into MySQL types (e.g., `String` to `VARCHAR`). Finally, it also assigns the UML class property upper cardinality value to the `column` tag `size` attribute.

```
[template public generateElement(e:Property)]
[if (e.upper>0)]
[if (not
  e.getModel().eContents(Class)->exists(l:Class |
  e.type=1))]
<column name="[e.name/]" [e.generateTipo()/] [if
  (e.isID)] primaryKey="true"[/if] [if
  (e.lower>0)]required="true"[/if]/> ... [/if]
```

**LISTING 2. Transformation rule generating table columns.**

#### 3) ONE-TO-ONE ASSOCIATIONS

The following ACCELEO code shows a fragment of the transformation rule that transforms mandatory one-to-one associations in UML class models into table foreign keys.

Note that the table that represents the class owning the relationship defines a column which name is the name assigned to the UML association in the UML class model. The name of class not owning the association with the prefix "pk" is used as column name if no name is assigned to the association.

#### 4) ONE-TO-MANY ASSOCIATIONS

The following ACCELEO code shows a fragment of the transformation rule that transforms one-to-many relationships in UML class models into table foreign keys. The approach to solve this pattern is analogous to the one employed in one-to-one relationships where the table attribute defining the relationship is derived from source and target class names if no association name is defined.

```
[if (not p.class.name.oclIsInvalid() and
  (p.upperBound()=1) and (p.class.name=e.name))]
<column name="fk[p.name/]" [if
  (p.lowerBound()>0)]required="true"[/if] [if
  (p.type.eAllContents(Property) ->
  select(l:Property | l.isID) -> size()=0)]
  type="INTEGER"[else] ... [/if] />
<foreign-key foreignTable="[p.type.name/]">
  <reference local="fk[p.name/]" foreign.../>
</foreign-key>[/if]
```

**LISTING 3. Transformation rule generating table foreign keys (one-to-one).**

```
[if (not p.class.name.oclIsInvalid() and
  (p.upperBound()<0 and
  p.opposite.upperBound()=1) and
  (p.type.name=e.name) and (p.class.name=e.name))]
<column name="fk[ass.memberEnd->at(2).name/]" [if
  (p.opposite.lowerBound()>0)]required="true"[/if]
  [if (p.class.eAllContents(Property)
->select(l:Property |
  l.isID)->size()=0)]type="INTEGER"[else]
[p.class.eAllContents(Property)->select(l:Property
  | l.isID)->asSequence()
->first().generateTipo()/][/if] />
<foreign-key foreignTable="[p.class.name/]">
  <reference
    local="fk[ass.memberEnd->at(2).name/]"
    foreign=[if
    (p.class.eAllContents(Property)->select(l:Property
    | l.isID)->size()=0)]"id[p.class.name/]"[else]
  "[p.class.eAllContents(Property)->select(l:Property
    |
    l.isID)->asSequence()->first().name/]"[/if]/>
</foreign-key>[/if]
```

**LISTING 4. Transformation rule generating table foreign keys (one-to-many).**

#### 5) MANY TO MANY ASSOCIATIONS

The following ACCELEO code shows a fragment of the transformation rule that transforms many-to-many relationships in UML class models into tables. In this case, the generated table links both tables; therefore, there is no need to add extra information on the generated tables representing classes.

### B. TEST CASES LAYER RULES

The test case layer rules generate the source code for case tests that check the code generated by Persistence layer rules.

The goal of test cases is the verification of the insert, read, update and delete database operations as well as the verification of association management among classes (i.e. one-to-one, one-to-many, and many-to-many).

These transformation rules in ACCELEO generate a test case file in PHP for each of class defined in the UML model. Because the process of generating test cases for primitive attributes is trivial, as an illustrative example, the following

```
[if (not p.class.name.oclIsInvalid() and
  (p.upperBound()<0 and
  p.opposite.upperBound()<0) and
  (p.class.name=e.name))]
[let nombreTabla2 : String = if
  (p.type.name.toString()<>p.class.name.toString())
  then 'id'.concat(p.type.name) else
  'relacion'.concat(p.name) endif]
<table name=[if
  (ass.name.oclIsUndefined())]"[p.class.name/]_
[p.name/]"[else]"[ass.name/]"[/if]
  isCrossRef="true">
 <column name="id[p.class.name/]"
   primaryKey="true"
 [if (p.class.eAllContents(Property)
->select(l:Property |
  l.isID)->size()=0)]type="INTEGER"[else]
[p.class.eAllContents(Property)
->select(l:Property |
  l.isID)->asSequence()->first().generateTipo()/]
[/if]/>
 <column name="[nombreTabla2/]" primaryKey="true"
 [if (p.type.eAllContents(Property)->select(
 l:Property |
   l.isID)->size()=0)]type="INTEGER"[else]
[p.type.eAllContents(Property)->select(l:Property
  |
l.isID)->asSequence()->first().generateTipo()/]
[/if]/>
 <foreign-key foreignTable="[p.class.name/]"
   name="[p.class.name/]_[p.name/]">
  <reference local="id[p.class.name/]" foreign=[if
    (p.class.getAllAttributes()
->select(l:Property |
  l.isID)->size()=0)]"id[p.class.name/]"[else]
"[p.class.getAllAttributes()->select(l:Property |
  l.isID)->asSequence()->first().name/]"[/if]/>
 </foreign-key>
<foreign-key foreignTable="[p.type.name/]"
  name="[nombreTabla2/]_[p.class.name/]">
 <reference local="[nombreTabla2/]" foreign=[if
   (p.type.eAllContents(Property)
->select(l:Property |
  l.isID)->size()=0)]"id[p.type.name/]"[else]
"[p.type.eAllContents(Property)->select(l:Property
  | l.isID)->asSequence()->first().name/]"[/if]/>
</foreign-key>
</table>
[/let][/if]
```

**LISTING 5.** Transformation rule generating tables and foreign keys (many).

code shows a fragment of the transformation rule that generates the unit test to verify the mappings of class properties resulting from UML class associations.

```
$b->add [claseAsiciacion.name.toUpperFirst()/]Many
[nombreTabla/] ([if (tipoPrimaria=null)] null
  [else] [tipoPrimaria/][/if])
[for (c:Property |
  claseAsociacion.eAllContents(Property) ->
  select(...))]
```

**LISTING 6.** Test case M2T transformation rule code generating test cases.

This transformation rule generates a function which name results from adding a prefix (i.e. add, remove,etc.) with the operation being tested, and postfix with the cardinality of the association to test. The way to generate these tests for the different types of relationships follows the same pattern.

### C. WEB SERVICE LAYER RULES

This section focuses on the M2T transformation rules in ACCELEO that generate the PHP Web service layer. This layer automatically connects the persistence layer code to the presentation layer code without the intervention of the application developers unless customization is required.

The following sections describe how the PHP classes implementing the system business logic are linked to the persistence layer implemented via Propel framework as well as how these classes are connected to the Web service layer implementing ReST API.

#### 1) BINDING BUSINESS LOGIC TO CRUD

The following code in ACCELEO shows a fragment of the transformation rule code that generates PHP classes that implement CRUD functions to access database information through the Persistence layer.

```
class [e.name.toUpperFirst()/] extends
  Base[e.name.toUpperFirst()/]{
  public function add[for (c:Generalization |
    e.generalization)]Gen[/for]($[clavePrimaria/]
    [for (c:Generalization | e.generalization)]
    [for (gen:Property |
    c.general.eAllContents(Property))]
    [gen.generateElementPHPArguments()/]
    [/for][/for] [for (c:Property |
    e.eAllContents(Property) -> select(l:Property
    | not l.isID))]
    [c.generateElementPHPArguments()/]
  [/for]){...
```

**LISTING 7.** Business logic to CRUD Web service M2T transformation rule.

The code generation is quite directly due to ReST architecture definition. This transformation rule generates a PHP file for each PHP class in the UML class model where the name of the generated PHP class corresponds to the class name in the UML model.

During the transformation process, each property defined in a class is transformed into a function parameter, including those defined in class hierarchies and one-to-one relationships. In one-to-one relationships, the name of each class property is prefixed with the name of the class it represents (if no name was assigned to the association). The one-to-many and many-to-many relationships are defined following an analogous process.

#### 2) BINDING BUSINESS LOGIC TO ReST API

The set of functions generated in the previous section are exposed as part of a SOA through a ReST API that employs different HTTP request methods (i.e., GET, POST, DELETE, and PUT) to read, insert, delete, and update database information using the persistence layer.

The matching among the different types of HTTP requests and corresponding functions is not generated through model transformations. Instead, we employ parsing techniques using both, request bodies and query strings from `$_POST`

and $_GET global variables. This mapping defined in the script.php file is unique and invariant for all requests.

### D. PRESENTATION LAYER RULES

The M2M transformation defined in ATL that generates the Presentation layer takes as input parameter the UML class model of the WebApp with the aim of generating a model in TagML [52] containing the representation of the XML documents that represent Polymer Web components.

The TagML DSL enables developers to capture the structure of tag-based documents in order to generate their code automatically, improving software generation re-usability and maintainability. The abstract syntax of the TagML DSL is summarized by the metamodel in Essential Model Object Facility (EMOF) [56] depicted in Figure 3.

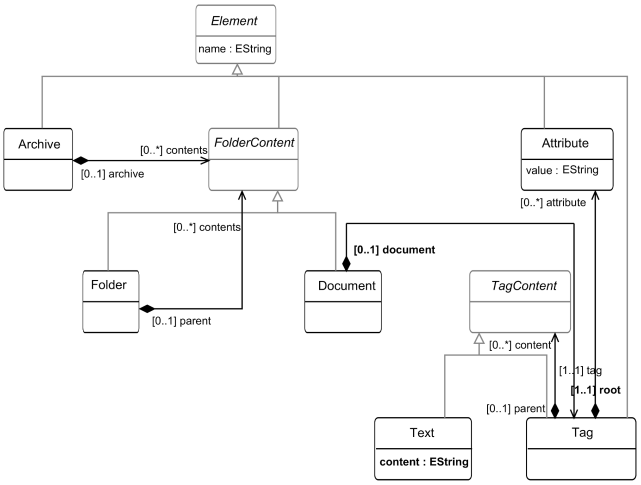


**FIGURE 3.** The metamodel in essential model object facility (EMOF) of the abstract syntax of the TagML domain specific language.

This metamodel captures tag names in the name attribute of Tag metaclass instances. The list of attributes of document tags are captured in the attributes Tag property, and nested tags or plain text are captured in the contains Tag tag property. Main advantage of employing this approach is decoupling tag based document semantic from syntax which enables developers to focus on the document structure instead of how to generate source code obtaining safer code [52].

#### 1) UML CLASSES TO WEB FORMS

This section focuses on how UML classes are transformed into Web forms. The following code is a fragment of the transformation rule that generates Web forms derived from a UML class model where the form tag is generated containing the result of applying the FormElementItem lazy rule that generates the set of fields based on class properties.

#### 2) UML CLASSES TO WEB NAVIGATION

For each class in the UML class model an option menu is generated. This menu enables users to view the list of class instances stored into the database, navigate through the class

```
lazy rule FormElement{
  from c: UML!Class, folder: TagML!Tag
  to ...
  form: TagML!Tag(
    name<-'form',
    parent <- divcard,
  ...
    content <- thisModule.getProperties(c)->
      collect(a| thisModule.FormElementItem(a,
      a.name+c.name, a.name, form)),
    attribute <- thisModule.Form(c.name.toLower(),
      form)
  ),...}
```

**LISTING 8.** Lazy rule generating TagML fields in TagML from UML properties.

instance associations, and insert or delete database records. This functionality is achieved applying the navigation transformation rule presented in the following excerpt of code where the FormElementItemView rule is called for each class to generate the set of buttons to navigate among class associations.

```
lazy rule VistaElement{
  from c: UML!Class, folder: TagML!Tag
  to ...
  form: TagML!Tag(
    name <- 'form',
    parent <- divcard,
    content <- thisModule.getProperties(c) ->
      collect(a |
      thisModule.FormElementItemView(a,form)),
    attribute <-
      thisModule.Form(c.name.toLower()+'action',form)
  ), ...)}

  lazy rule FormElementItemView{
  from c: UML!Property, form: TagML!Tag to ...
  do{
    if(c.upper>0 and not
      c.class.getModel().eContents()->select(t |
      t.oclIsTypeOf(UML!Class))->exists(l|
      c.type=l) ){  ... } else {
      input.name<-'null';
    if (c.upperBound()=1 and c.lowerBound()=1 and
      not c.type.name.oclIsUndefined()){
     form.content<-
       thisModule.ButtonForeignInformation(c.type,
       form);
    }
    if (c.upperBound()<0 and
      c.opposite.upperBound()=1 and not
      c.type.name.oclIsUndefined()){
    form.content <-
      thisModule.ButtonForeignInformation(
      c.association.memberEnd -> at(2), form);
    }...}}}
```

**LISTING 9.** Lazy rules that generating UI main navigation menu.

#### 3) MANAGEMENT OF UML CLASS ASSOCIATIONS

The UML class association management is based on providing users with the ability to associate and disassociate class instances. The following code excerpt is part of the VistaElement transformation rule which shows how associations are detected and how the BlockElementSelect transformation rule generates the panel within the UI responsible for associating and disassociating class instances.

```
lazy rule VistaElement{
  from c: UML!Class, folder: TagML!Tag
  to ...
  do {
    for (ass in c.getAssociations()){
    for (p in ass.memberEnd){
      if(not p.class.oclIsUndefined()){
      if ((p.upperBound() < 0 and
        p.opposite.upperBound() < 0) and not
        ((p.type.name=c.name) and (p.class.name =
        p.type.name))){
        if (p.class.name <> c.name){...}
        else{...}
      }
      if ((p.upperBound() < 0 and
        p.opposite.upperBound() < 0) and
        (p.type.name=c.name) and
        (p.class.name=p.type.name)){...}
      if ((p.upperBound() < 0 and
        p.opposite.upperBound() = 1) and
        (p.type.name = c.name) and (p.class.name =
        c.name)){...}
      if (p.upperBound() = 1 and
        p.opposite.upperBound() < 0 and
        (p.type.name = c.name) and (p.class.name
        <> p.type.name)){...}
  }}}}}
```

**LISTING 10.** Lazy rules generating UI association management UIs.

## V. CASE STUDY: APPLICATION OF THE TRANSFORMATION ARCHITECTURE

This section presents The Travel Management WebApp case study generated with the proposed model transformation architecture where users are capable of organizing/managing tour trips. The goal of the system is the management of groups of users who travel to a common destination using a travel agency.

We firstly present the generation of the WebApp described in Section III and IV employing the proposed transformation architecture in the first subsection and then the execution of a concrete task in the second subsection.

The development process depicted in Figure 1 is defined in terms of the execution of five model transformations (i.e. Persistence M2T, Test Case M2T, Web Service M2T, Presentation M2M, and Presentation M2T) resulting in the generation of the source code of WebApps distributed in three different server nodes (i.e. Presentation, Web Service, and Database servers).

### A. THE DEVELOPMENT PROCESS

The following paragraphs describe the development process followed to generate The Travel Management WebApp using the proposed model transformation architecture. As depicted in Figure 1, the domain model definition is the starting point of the development process, and the order of the transformation execution is independent of the result, except for the Presentation M2M and M2T transformations that should be performed sequentially.

#### 1) DOMAIN MODEL DEFINITION

The development process starts with the definition of the WebApp UML class diagram depicted in Figure 4, which might be defined with any CASE tool following OMG MDA standards. In our proposal we use Papyrus Eclipse IDE distribution for the creation of the UML Class model, depicted in Figure 4, which is the input parameter of the model transformation architecture.

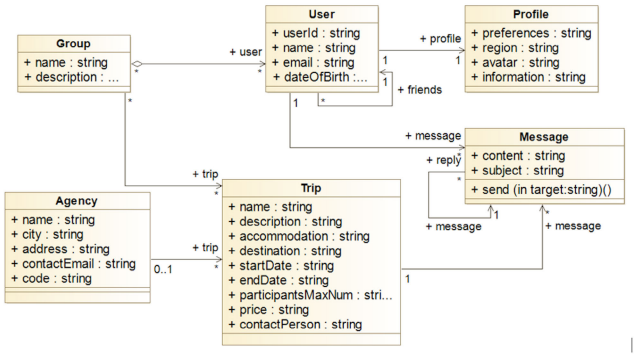


**FIGURE 4.** The UML class diagram corresponding to the travel management case study.

#### 2) PERSISTENCE M2T TRANSFORMATION

The result of the Persistence M2T transformation execution taking the UML class model as input parameter generates two files defining the database schema file and the object to relational database mapping file which are employed by Propel framework scripts to define the database structure and the set of PHP class files that implement the application business logic and data access. Consequently, this transformation generates the Persistence layer source code which is distributed between the Database server (database schema definition) and the Web Service Server (PHP classes to access database information).

The persistence layer employs the Propel framework to manage relational database persistence in MySQL. Propel defines relational database schema in the `schema.xml` file which is generated by the Persistence M2T transformation.

The following excerpt of the `schema.xml` file code shows the result of applying the transformation rule that derives database table schemes from UML classes (see Section IV-A1 for details). As result, this transformation rule assigns `User` to the `name` and `phpName` attributes of the `table` tag. In addition, as `User` class is not abstract, it does not set the `abstract` attribute to `true`.

```
<table name="User" phpName="User"
  allowPkInsert="true">
```

**LISTING 11.** Result UML class to table rule to the user class.

The result of generating table attribute definitions in Propel `schema.xml` file applying the rule described in Section IV-A2 to the `email` attribute is presented in the following XML excerpt of code where the `name`, `type`, and `size` attributes of the `column` tag are set to `email`, `VARCHAR`, and `100` according to the `email` UML class property definition of the `User` class.

```
<column name="email" type="VARCHAR" size="100" ...
```

**LISTING 12.** Result of class property rule to user email property.

The next excerpt of the `schema.xml` file shows the result of the transformation rule described in Section IV-A3 to the association between `User` y `Profile` which generates fkProfile table foreign key in `User` table linking `User` to `Profile` using `idProfile` primary key. Note that the table attribute is defined as mandatory because the association defines a cardinality greater than 1.

```
<column name="fkProfile" required="true"
  type="INTEGER" />
<foreign-key foreignTable="Profile">
 <reference local=""fkProfile"␣
   foreign="idProfile"/>
</foreign-key>
```

**LISTING 13.** Result of one-to-one rule to user-profile association.

The following fragment of code shows the result of applying the one-to-many Persistence M2T transformation rule described in Section IV-A4 to the UML association between `Trip` and `Message` classes where the `fkTrip` foreign key is generated in `Message` table to link `Message` class instances to a `Trip` class instance using `idTrip` primary key. The approach to solve this pattern is analogous to the pattern used in one-to-one relationships.

```
<column name="fkTrip" type="INTEGER" />
<foreign-key foreignTable="Trip">
 <reference local="fkTrip" foreign="idTrip"/>
</foreign-key>e local=""fkProfile"␣
  foreign="idProfile"/>
</foreign-key>
```

**LISTING 14.** Result of one-to-many rule to trip-message association.

The result of applying the many-to-many association Persistence M2T transformation rule described in Section IV-A5 to the association between `Trip` and `Group` UML classes is presented in the following excerpt of XML code where the `Group_Trip` table is created to relate `Trip` and `Group` tables using `idTrip` and `idGroup` attributes to link `idTrip` and `idGroup` primary keys of `Trip` and `Group` tables respectively.

```
<table name="Group_Trip" isCrossRef="true">
 <column name="idGroup" primaryKey="true"
   type="INTEGER"/>
 <column name="idTrip" primaryKey="true"
   type="INTEGER"/>
 <foreign-key foreignTable="Group"
   name="Group_Trip">
  <reference local="idGroup" foreign="idGroup"/>
 </foreign-key>
 <foreign-key foreignTable="Trip"
   name="idTrip_Group">
  <reference local="idTrip" foreign="idTrip"/>
 </foreign-key>
</table>
```

**LISTING 15.** Result of many-to-many rule to trip-group.

### 3) TEST CASES M2T TRANSFORMATION

The execution of the Test Case M2T transformation generates the set of test cases that check the database access functionality through the PHP classes generated during the Persistence M2T transformation execution. As this code is intended to be used during the application development, it is not located at any production server node. However, it requires the classes generated by the Persistence M2T transformation to access database information.

As mentioned in Section IV-B, the result of generating test cases for primitive type attributes is trivial; therefore, we focus the explanation on the generation of test cases for UML associations. The following fragment of PHP code for the PHPUnit framework is the result of applying the Test cases M2T transformation rule to the association between the `Trip` and `Agency` classes.

```
$b->addTripMany(null, "cadenaprueba", "This␣is␣a␣
  test␣string", "This␣is␣a␣test␣string" ... )
```

**LISTING 16.** Result of test cases rule trip-agency association.

This transformation rule generates the `addTripMany` function to check the `addTrip` function of `Agency`.

### 4) WEB SERVICE M2T TRANSFORMATION

The result of executing the Web Service M2T transformation taking also the UML class model as input parameter generates the `script.php` file which is in charge of processing ReST API requests (e.g. `POST/api/trip`) using PHP classes generated with the Persistence M2T transformation to implement WebApp business logic. This transformation generated the source code for the Web Service Layer which is located in the Web Service server node.

One of the goals of the Web Service layer of the WebApp is the binding of the business logic to the CRUD operations on the database as exposed in Section IV-C1.

The following excerpt of code shows a fragment of the `User` PHP class generated from the UML class model. It shows how the name of the `User` UML class model is mapped to the name of the PHP class which extends the `UserBase` PHP class that defines the set of functions related to the persistence layer. In addition, properties, such as `$idUser` and those related to `$one-to-one` associations, are mapped to function parameters; for instance, the `Profile` UML class defines `$gustosProfile` attribute.

```
class User extends BaseUser{
  public function add($idUser, $Profile, ...
  , $gustosProfile, $residenciaProfile ... ) ...
```

**LISTING 17.** Result of web service rule one-to-one user-profile association.

The other goal of the Web service layer is binding WebApp business logic to the ReST API. As mentioned in Section IV-C2 no transformation is required because the binding is defined via pattern matching.

#### 5) PRESENTATION M2M AND M2T TRANSFORMATIONS

Finally, the execution of the Presentation layer source code requires the execution of the Presentation M2M and Presentation M2T transformations. While the Presentation M2M transformation execution generates the model of the XHTML documents containing CSS and JavaScript references; the Presentation M2T transformation execution generates the source code of the Presentation layer which is located in the Presentation server node.

The application of the Presentation M2M and M2T transformation rules to the class UML model generates the WebApp presentation layer for the Polymer framework exposed in Section IV-D.

The result of applying the class to Web form transformation rules described in Section IV-D1 taking the `User` UML class as input parameter is depicted in Figure 5.

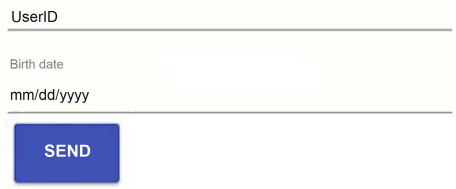


**FIGURE 5.** The polymer web component generated using the `FormElement` transformation rule applied to the `User` UML class.

The WebApp navigation is generated using the transformation rules defined in Section IV-D2. The result of applying these rules to the `User` UML class model is shown in Figure 6.

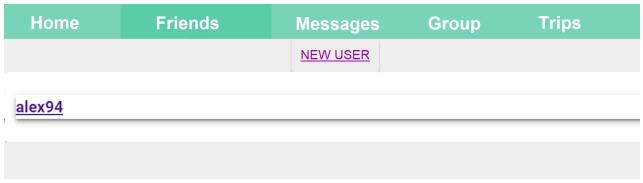


**FIGURE 6.** The polymer web component generated using the `VistaElement` transformation rule applied to the `User` UML class.

The transformation definition that generates the UI capable of associating and disassociating class instances is explained in Section IV-D3. The result of applying these transformation rules is depicted in Figure 7 and Figure 8, which show the UIs generated for the many-to-many relationship between `Trip` and `Agency` UML classes. While Figure 7 depicts the panel to manage instance associations from the `Trip` UI, Figure 8 depicts the panel to manage instance associations from the `Agency` UI.

### B. THE "ADD TRIP" TASK PERFORMANCE

This section explains how the "Add Trip" task is performed through the multi-layer WebApp generated by the model transformation architecture. Figure 9 shows an overview of the execution sequence for "Add trip" task where users are

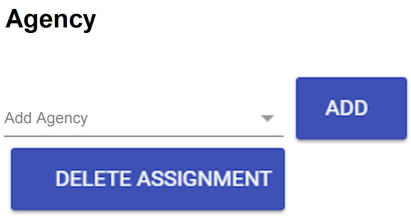


**FIGURE 7.** The UI panel from the trip view.


**FIGURE 8.** The UI panel from the agency view.

capable of publishing a trip showing the relationship between Presentation, Web Service, and Database servers.

Under this scenario, users access The Travel Management WebApp home page in the Presentation Server. To perform the "Add Trip" task, they access the list of trips by clicking on the Trip menu option on the left. As result of this action, users access the list of Trips which is retrieved from the Web Service server performing a GET request on the Web Service server which processes the request using the Persistence layer PHP code to access database information using SQL SELECT.

To add a new trip, users click on the NEW TRIP button to access the Web form to introduce the trip information. Once the Web form information is set, users click on the SEND button to POST the Web form information in JSON format as part of the request body to the Web Service server. The Web Service server processes the request using PHP Persistence layer PHP code to perform an SQL INSERT command on the database. As a result of this action, users are redirected to the list of trips page retrieving the list of trips including the new trip.

### C. QUALITATIVE EVALUATION

This section exposes a qualitative evaluation of the results of applying the proposed transformation architecture development process to the case study.

Table 1 presents the results of running the process proposed in Section III applying the transformation rules to the case study which UML class diagram is depicted in Figure 4.

From the efficiency point of view, once the model transformations are implemented, the applications development time is limited to UML class diagram design because it is incomparable to the time required by the code generation process.

Assuming model transformations are fully implemented and tested, the effectiveness of the generated Web application is higher compared to a traditional development process because it is less prone to errors. The main reason behind

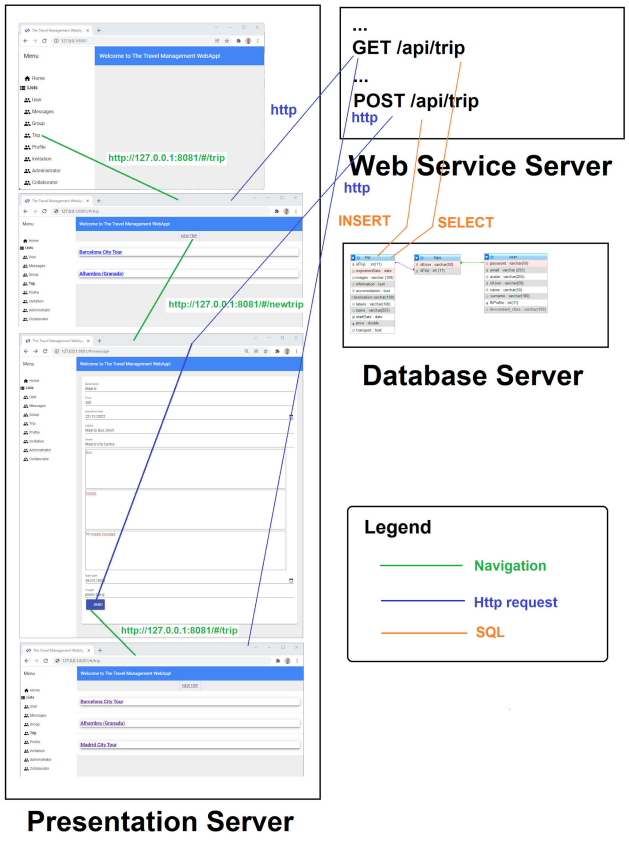


**FIGURE 9.** The overview of the "add trip" task performed on the multi-layer web application generated with the proposed model transformation architecture.

**TABLE 1.** The travel management WebApp case study software artifacts generation.

| Software artifact | Quantity |
|---|---|
| Database tables (persistence layer) | 10 |
| Web service layer PHP files | 6 |
| Web service layer functions | 60 |
| Persistence unit test PHP files | 6 |
| Persistence unit test layer unit tests | 31 |
| Presentation layer data-bindings | 14 |
| Presentation layer lists | 11 |
| Presentation layer Web components | 25 |

this statement is the lack of developer interventions in the process because applications developed using this proposal, are automatically generated. Moreover, transformation corrections are automatically propagated to both, past and future developments due to automatic application regeneration.

Finally, this proposal improves the UI properties such as consistency and homogeneity since automatic code generation enable UI generation in a systematic way, avoiding the inconsistencies inherited from the traditional manual development process.

## VI. DISCUSSION

The novelty of our approach is the definition of a model transformation architecture capable of generating full stack WebApps source code at three different layers (persistence, web service, and presentation) and additionally, test cases; unlike most proposals which address one or two layers (e.g. persistence, web service and presentation, only presentation, etc.). In addition, our proposal provides a set of tools (i.e. Eclipse plugins) capable of managing model transformations and multi-layer integration which is also inter-operable with third party tools following OMG MDA standards.

The MDD has been extensively explored in the software engineering research area. The authors of [61] present a systematic literature review (SLR) between 2009 and 2019 analyzing MDD proposals that generate source code.

As a result of this SLR, they state that UML is the most commonly used modeling language to represent software models. It validates the widely use of UML class models to model the application to be developed unlike other proposals that use other languages (e.g. WebML [62] is used in [63] to create Web application models). The advantages of using a popular and well-known modeling language are the ease of learning this language and the reuse tools and design patterns, which significantly reduce the applications design time [64]. The SLR also highlights that the most used abstraction layers in MDA that generate source code are the PIM and PSM layers, which are the ones that are implemented in this proposal.

The analysis of existing proposals performed in section II shows the main shortcomings identified. Table 2 shows a comparison of the proposals analysed regarding the languages used, whether they comply with international standards or not, and the layers they generate.

At the end of the table, we show the features corresponding to the proposed solution. As we can see, most proposals only generate one or two layers whereas our approach covers the three layers. In addition, it is based on OMG MDA standard, as many others, and uses UML as main language. Some of the proposals are limited to the generation of the presentation layer source code only. They do not generate any source code for the data access or Web services layers. Unlike this proposal, they lack of a comprehensive integrated solution does not enable the automatic generation of the association between layers.

In summary, the main shortcomings found in existing proposals are that none of them addresses the generation of the three layers, others does not comply with international standards, and others lack of multi-layer integration. Other of the main limitations of existing proposals compared to the one presented in this article is the lack of interoperability at both, design-time and run-time. The proposal we present goes beyond the current state of the art by addressing these shortcomings in an integrated way, with the generation of the three layers: presentation, web service and persistence, and an additional test case layer,following a well-defined model transformation architecture.

Among the limitations of the proposal, we can highlight the lack of a customization mechanism for the user interface labels generated, as it takes the labels from the Class diagram.

**TABLE 2.** Comparison of existing proposals.

| Proposals | Languages | Standard Compliance | Presentation Layer | Web Service Layer | Test Case Layer | Persistence Layer |
|---|---|---|---|---|---|---|
| [21] | WSDL | W3C | * | * | | |
| [22] | WADL | W3C | * | * | | |
| [25] | WSDL | MDA | * | * | | |
| [27] | UML Profiles | MDA | * | | | |
| [28] | UML Profiles/Java | MDA | * | | | * |
| [29] | UML Profiles/Java | MDA | * | * | | |
| [30] | UML Profiles | MDA | * | * | | |
| [31] | UML Profiles | MDA | * | * | | |
| [32] | UML Profiles/AngularJS | MDA | * | ? | | |
| [33] | WS-BPEL | W3C | ? | * | | |
| [34] | Spreadsheet | Monolithic | | | | |
| [35] | UML | MDA | Android | | | |
| [37] | RIA | | * | | | |
| [38] | | MDA | Android | | | |
| [39] | DSL | MDA | * | | | |
| [43] | DSL | MDA | * | | | |
| [41] | DSL | MDA | * | | | |
| [44] | DSL | MDA | * | | | * |
| [45] | ASP.NET | | * | | | * |
| [46] | UML/XML/PHP | W3C/MDA | * | | | * |
| [48] | UML | MDA | * | | | * |
| Proposed Solution | UML | MDA | * | * | * | * |

Another limitation is that the generation of the Persistence Layer currently only supports the Relational Database Management System (DBMS), concretely, MySQL. In addition, the generation of the Web service layer has been developed to support only ReST services using JSON.

The last limitation regards the generation of the Presentation layer, which only supports Polymer and JavaScript.

## VII. CONCLUSION

This proposal defines a MDA based transformation architecture that supports the MDD framework for the automatic generation of three-layer WebApps (i.e., Persistence, Web services, and Presentation layers) following OMG standards. To meet this goal, a set of three M2T transformations is presented in ACCELEO as well as a M2M transformation in ATL that guide the development process using a single UML class model as input parameter.

The generated applications employ a client-server architecture where the server-side of the system is in PHP and client-side in HTML and JavaScript. The resulting applications implement a persistence layer with the Propel framework as an Object Relational Mapping (ORM) broker for the MySQL DBMS. In addition, these applications are equipped with a Web service layer that defines an API following a ReST service architecture using JSON. This layer communicates with the Persistence layer so that the generated application has a high interoperability capability with third-party run-time systems. In turn, the generated applications define a presentation layer implemented with the Polymer framework that connects to the Web service layer to access the system database.

From a development point of view, the set of tools presented in this proposal follows the OMG MDA standards ensuring a great interoperability at design-time as well as and optimal integration with third-party tools. As a demonstration of the integration capability of the developed tools, they are integrated into the Eclipse integrated development environment using the Eclipse Papyrus plugin to define the UML class model representing the application to be developed and used as input parameter of model transformations.

Another issue that demonstrates the integration capability of the proposal is the use of a M2M transformation that generates a TagML model, rather than a single M2T transformation that generates text, simplifying the XML document generation process while ensuring its robustness.

A case study presenting how the proposed model transformation architecture is used to validate the proposal and develop the multi-layer Travel Agency Management WebApp. This case study shows how the transformation rules generate the WebApp source code using a UML Class model as a transformation input parameter. In addition, this case study presents how the generated source code interact with the Persistence, Web Service and Presentation layers of the application to perform the ''Add Trip'' task.

Finally, several research lines are open as future works. The first of them is the development of model transformations to enable the use of different technologies associated to different application layers. For example, in the persistence layer, we consider the definition of a transformation to generate the source code required to replace relational database management system by a non-relational database manager (NoSQL) such as MongodDB.

We are also exploring the ability to generate a SOAP-based Web services layer that employs XML, instead of JSON, to improve the system interoperability at run-time, e.g. to communicate with legacy systems. Moreover, we are

considering the development of model transformations that use different technologies for the presentation layer (e.g. AngularJS). In addition, we are also exploring the feasibility of generating source code for different mobile platforms, such as Android or iOS.

## REFERENCES

[1] R. T. Fielding, "ReST: Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.
[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. (2000). *Simple Object Access Protocol (SOAP) 1.1*. Accessed: Dec. 01, 2021. [Online]. Available: http://www.w3.org/TR/soap
[3] *Object Management Group (OMG)*. Accessed: Dec. 01, 2021. [Online]. Available: http://www.omg.org
[4] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise, *MDA Distilled*. Reading, MA, USA: Addison-Wesley, 2004.
[5] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed. London, U.K.: Pearson, 2004.
[6] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Reading, MA, USA: Addison-Wesley, 2003.
[7] Oracle. *MySQL*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.mysql.com/
[8] *PostgreSQL*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.postgresql.org/
[9] Microsoft. *SQLServer*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.microsoft.com/es-es/sql-server
[10] *MongoDB*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.mongodb.com
[11] S. Bergmann. *PHPUnit*. Accessed: Dec. 01, 2021. [Online]. Available: https://phpunit.de/
[12] OpenJS. *Mocha*. Accessed: Dec. 01, 2021. [Online]. Available: https://mochajs.org/
[13] K.-M. Chung, *JavaServer Pages 2.3 Specification*. Santa Clara, CA, USA: Oracle. 2013.
[14] *PHP*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.php.net/
[15] E. Burns and M. Riem, *JavaServer Faces Specification*. Santa Clara, CA, USA: Oracle, 2017.
[16] Google. *Angular JS*. Accessed Dec. 01, 2021. [Online]. Available: https://angular.io/
[17] E. You. *Vue.js The Progressive Javascript Framework*. Accessed: Dec. 01, 2021. [Online]. Available: https://vuejs.org/
[18] Google. *Material*. Accessed: Dec. 01, 2021. [Online]. Available: https://material.io/
[19] TP Group. *Polymer*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.polymer-project.org/
[20] Eclipse. *Eclipse IDE*. Accessed: Dec. 01, 2021. [Online]. Available: http://www.eclipse.org
[21] M. Kassoff, D. Kato, and W. Mohsin, "Creating GUIs for web services," *IEEE Internet Comput.*, vol. 7, no. 5, pp. 66–73, Sep. 2003.
[22] J. Spillner, M. Feldmann, I. Braun, T. Springer, and A. Schill, "Ad-hoc usage of web services with Dynvoker," in *ServiceWave: Towards a Service-Based Internet* (Lecture Notes in Computer Science), vol. 5377, P. Mähönen, K. Pohl, and T. Priol, Eds. Berlin, Germany: Springer, 2008, pp. 208–219.
[23] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001). *Web Services Description language (WSDL) 1.1, W3C*, Accessed: Dec. 01, 2021. [Online]. Available: http://www.w3.org/TR/wsdl
[24] M. Hadley, "Web application description language (WADL)," Sun Microsystems, Santa Clara, CA, USA, Tech. Rep. TR-2006-153, Apr. 2006.
[25] M. Feldmann, G. Hubsch, T. Springer, and A. Schill, "Improving task-driven software development approaches for creating service-based interactive applications by using annotated web services," in *Proc. 5th Int. Conf. Next Gener. Web Services Practices*, Sep. 2009, pp. 94–97.
[26] F. P. Marzullo, J. M. D. Souza, and J. R. Blaschek, "A domain-driven development approach for enterprise applications, using MDA, SOA and web services," in *Proc. 10th IEEE Conf. E-Commerce Technol., 5th IEEE Conf. Enterprise Comput., E-Commerce E-Services*, Jul. 2008, pp. 432–437.
[27] S. A. Mubin and A. H. Jantan, "A UML 2.0 profile web design framework for modeling complex web application," in *Proc. 6th Int. Conf. Inf. Technol. Multimedia*, Nov. 2014, pp. 324–329.
[28] Y.-C. Huang, C.-P. Chu, Z.-A. Lin, and M. Matuschek, "Transformation from web PSM to code," in *Proc. Int. Conf. Distrib. Multimedia Syst. (DMS)*, 2009, pp. 1–4.
[29] M. Kataria, R. Yadav, and A. Khunteta, "A component-centric UML based approach for modeling the architecture of web applications," *Int. J. Recent Res. Rev.*, vol. 5, pp. 22–27, Mar. 2013.
[30] I.-C. Hsu, "Visual modeling for web 2.0 applications using model driven architecture approach," *Simul. Model. Pract. Theory*, vol. 31, pp. 63–76, Feb. 2013.
[31] S. Kaewkao and T. Senivongse, "A model-driven development of web-based applications on Google app engine platform," in *Proc. 10th Nat. Conf. Comput. Inf. Technol. (NCCIT)*, 2014, pp. 140–145.
[32] W. Chansuwath and T. Senivongse, "A model-driven development of web applications using AngularJS framework," in *Proc. IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2016, pp. 1–6.
[33] V. D. Pillai, "Development of a novel software architecture for active internet applications based on fusion of mobile agent, web services and BPEL technologies," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2010, pp. 652–653.
[34] K. Shiohara and X. Chen, "A concept of extending spreadsheet cell functions for web application development based on a cloud platform," in *Proc. IEEE Workshop Adv. Res. Technol. Ind. Appl. (WARTIA)*, Sep. 2014, pp. 1362–1365.
[35] A. Sabraoui, M. E. Koutbi, and I. Khriss, "GUI code generation for Android applications using a MDA approach," in *Proc. IEEE Int. Conf. Complex Syst. (ICCS)*, Nov. 2012, pp. 1–6.
[36] M. Lettner and M. Tschernuth, "Applied MDA for embedded devices: Software design and code generation for a low-cost mobile phone," in *Proc. IEEE 34th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2010, pp. 63–68.
[37] S. Roubi, M. Erramdani, and S. Mbarki, "Modeling and generating graphical user interface for MVC rich internet application using a model driven approach," in *Proc. Int. Conf. Inf. Technol. Organizations Develop. (IT4OD)*, Mar. 2016, pp. 1–6.
[38] J. D. Monte-Mor, E. O. Ferreira, H. F. Campos, A. M. da Cunha, and L. A. V. Dias, "Applying MDA approach to create graphical user interfaces," in *Proc. 8th Int. Conf. Inf. Technol., New Gener.*, Apr. 2011, pp. 766–771.
[39] Z. Morales, C. Magaña, J. A. Aguilar, A. Zaldívar-Colado, C. Tripp-Barba, S. Misra, O. Garcia, and E. Zurita, "A baseline domain specific language proposal for model-driven web engineering code generation," in *Proc. Comput. Sci. Appl. (ICCSA)*, O. Gervasi, B. Murgante, S. Misra, A. M. A. Rocha, C. M. Torre, D. Taniar, B. O. Apduhan, E. Stankova, S. Wang, Eds. New York, NY, USA: Springer, 2016, pp. 50–59.
[40] G. Sebastian, R. Tesoriero, and J. A. Gallud, "Modeling language-learning applications," *IEEE Latin Amer. Trans.*, vol. 15, no. 9, pp. 1771–1776, Aug. 2017.
[41] G. Sebastian, R. Tesoriero, and J. A. Gallud, "Automatic code generation for language-learning applications," *IEEE Latin Amer. Trans.*, vol. 18, no. 8, pp. 1433–1440, Aug. 2020.
[42] R. Tesoriero, "Distributing user interfaces," in *Proc. Workshop Distrib. User Interfaces Multimodal Interact.* Toulouse, France: ACM, 2014, pp. 1–10.
[43] R. Tesoriero and A. H. Altalhi, "Model-based development of distributable user interfaces," *Universal Access Inf. Soc.*, vol. 18, no. 4, pp. 719–746, Nov. 2019.
[44] G. Sebastián Rivera, R. Tesoriero, and J. A. Gallud, "Model-based approach to develop learning exercises in language-learning applications," *IET Softw.*, vol. 12, no. 3, pp. 206–214, Jun. 2018.
[45] H. Benouda, M. Azizi, M. Moussaoui, and R. Esbai, "Automatic code generation within MDA approach for cross-platform mobiles apps," in *Proc. 1st Int. Conf. Embedded Distrib. Syst. (EDiS)*, Dec. 2017, pp. 1–5.
[46] O. Betari, M. Erramdani, S. Roubi, K. Arrhioui, and S. Mbarki, "Model transformations in the MOF meta-modeling architecture: From UML to CodeIgniter PHP framework," in *Proc. Eur. MENA Cooperation Adv. Inf. Commun. Technol.*, Á. Rocha, M. Serrhini, and C. Felgueiras, Eds. New York, NY, USA: Springer, 2017, pp. 227–234.
[47] *CodeIgniter*. Accessed: Dec. 01, 2021. [Online]. Available: https://codeigniter.com/

[48] M. Rahmouni and S. Mbarki, "Model-driven generation: From models to MVC2 web applications," *Int. J. Softw. Eng. Appl.*, vol. 8, no. 7, pp. 73–94, Jul. 2014.
[49] Eclipse. *ACCELEO Transformation Language*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.eclipse.org/acceleo/
[50] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, nos. 1–2, pp. 31–39, 2008.
[51] Eclipse. *ATLAS Transformation Language*. Accessed: Dec. 01, 2021. [Online]. Available: https://www.eclipse.org/atl/
[52] R. Tesoriero, G. Sebastian, and J. A. Gallud, "TagML—An implementation specific model to generate tag-based documents," *Electronics*, vol. 9, no. 7, p. 1097, Jul. 2020, doi: 10.3390/electronics9071097.
[53] Eclipse. *Papyrus*. Accessed: Dec. 01, 2021. [Online]. Available: https://eclipse.org/papyrus/
[54] *OMG Unified Modeling Language (OMG UML), Superstructure, Ver. 2.4.1*, Object Management Group, Needham, MA, USA, Aug. 2011.
[55] *XML Metadata Interchange (XMI) Specification Ver. 2.5.1*, Object Management Group, Needham, MA, USA, Jun. 2015.
[56] *Meta Object Facility (MOF) Core Specification. Ver. 2.5.1*, Object Management Group, Needham, MA, USA, Oct. 2019.
[57] F. Zaninotto, W. Durand, H. Hamon, M. J. Schmidt, J. Augustin, T. Uebernickel, C. Cinotti, R. Dupret, M. Staab, and M. Scholten. *Propel*. Accessed: Dec. 01, 2021. [Online]. Available: http://propelorm.org/
[58] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC2616: Hypertext transfer protocol–HTTP/1.1," Tech. Rep. 2616, 1999.
[59] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Reading, MA, USA: Addison-Wesley, 1994.
[60] *Bower*. Accessed: Dec. 01, 2021. [Online]. Available: https://bower.io/
[61] G. Sebastián, J. A. Gallud, and R. Tesoriero, "Code generation using model driven architecture: A systematic mapping study," *J. Comput. Lang.*, vol. 56, Feb. 2020, Art. no. 100935.
[62] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (WebML): A modeling language for designing web sites," *Comput. Netw.*, vol. 33, nos. 1–6, pp. 137–157, Jun. 2000.
[63] G. Zhuang and J. Du, "MDA-based modeling and implementation of E-commerce web applications in WebML," in *Proc. 2nd Int. Workshop Comput. Sci. Eng.*, vol. 2, 2009, pp. 507–510.
[64] H. M. Fardoun, R. Tesoriero, G. Sebastian, and N. Safa, "A simplified MbUID process to generate web form-based UIs," in *Proc. 13th Int. Conf. Softw. Technol.* Setúbal, Portugal: SciTePress, 2018, pp. 801–808.

**RICARDO TESORIERO** received the Ph.D. degree from the University of Castilla-La Mancha (UCLM), Spain. He performed a postdoctoral stay at the Universitè Catholique de Louvain (UCL), Louvain-La-Neuve, Belgium, where he conducted research activities within the model-driven development of user interfaces research field. He has published more than 70 research articles in international conferences and journals. He has participated in several scientific committees of international conferences and workshops, including Distributed User Interfaces (DUI), Interaction, ISEC, and IADIS/WWW (La Web). His main research interests include the model-driven development of user interfaces and context-aware applications in ubiquitous computing environments.

**ALEJANDRO RUEDA** received the degree in computer science and engineering from the University of Castilla-La Mancha, Albacete, Spain, in 2017. Currently, he is an Application Security Analyst at Deloitte, Madrid, Spain. His main research interests include the development of web applications, including both front-end and back-end as well as the use of model-driven development of web applications.

**JOSE A. GALLUD** received the M.Sc. degree in computer science from the University of Murcia and the Ph.D. degree in computer science from the Polytechnic University of Valencia. He is an Associate Professor at the University of Castilla-La Mancha. He co-leads the Interactive Systems Engineering (ISE) Research Group, Albacete Research Institute of Informatics. His main research interests include human–computer interaction, software engineering, development of interactive systems, and distributed user interfaces. He has published more than 100 scientific papers in these areas, disseminated in journals, and international conferences books and chapters. He has participated in the organization of numerous national and international conferences in these research areas as the technical program chair, the Technical Program Member, and an Organizing Committee Member.

**MARIA D. LOZANO** received the M.Sc. and Ph.D. degrees in computer science from the Polytechnic University of Valencia, Spain. She has been an Associate Professor at the University of Castilla-La Mancha, since 2003. She co-leads the Interactive System Engineering (ISE) Research Group, Albacete Research Institute of Informatics. She is an author of more than 100 papers in indexed journals and international conference. Her teaching and research interests include software engineering and human–computer interaction. Her research interests include natural user interfaces, tangible user interfaces, affective computing, and user experience. She has served as the chair and a member for different program committees of national and international conferences.

**ANIL FERNANDO** (Senior Member, IEEE) received the B.Sc. degree (Hons.) in electronics and telecommunications engineering from the University of Moratuwa, Sri Lanka, in 1995, the M.Sc. degree (Hons.) in telecommunications from the Asian Institute of Technology (AIT), Thailand, in 1997, and the Ph.D. degree in video coding from the Department of Electrical and Electronic Engineering, University of Bristol, U.K., in 2001. He was a Reader with the University of Surrey, U.K.; a Senior Lecturer with Brunel University London, U.K., and an Assistant Professor with the AIT. He is currently a Professor of video coding and communications at the University of Strathclyde, Glasgow, U.K., and a Visiting Professor at the University of Surrey. He has authored over 350 international publications in video coding, machine learning, communications, and signal processing. His research interests include video coding and communications, machine learning and AI solutions for industrial applications, quality of experience modeling, autonomous systems, and 5G/6G application developments. He is a fellow of the Higher Education Academy, U.K., and a member of EPSRC College, U.K.