MVCLang: A Software Modeling Language for the Model-View-Controller Design Pattern

Mert Ozkaya[1] and Irem Fidandan[2]
[1]*Department of Computer Engineering, Yeditepe University, Istanbul, Turkey*
[2]*Eryaz Software, Istanbul, Turkey*

Keywords: Model-View-Controller (MVC), Software Modeling Language, Model Analysis, Code Generation, ASP.NET.

Abstract: The Model-View-Controller (MVC) software design pattern promotes separating software systems into the model, view, and controller elements. The views represent the user-interfaces, the models represent the system data, and the controllers handle the requests sent by the views and coordinate the interactions between views and models. While many software frameworks are available for the MVC-based software developments, no any attempt have been made on increasing the level of abstraction for the MVC developments and provide a model-based approach. Indeed, none of the high-level software modeling languages support the MVC design pattern. So, we propose in this paper a visual, MVC-based modeling language called MVCLang, which enables to model MVC-based software architectures that can be easily analysed and implemented. MVCLang is supported with an Eclipse-based prototype toolset for specifying the visual MVC architectures and analysing them for a number of wellformedness rules. MVCLang's toolset can further produce ASP.NET MVC code that reflects the architectural design decisions. We evaluated MVCLang on a software company that offers e-commerce solutions. Therein, 5 developers used MVCLang for their e-commerce project developments and provided feedback for a set of pre-determined questions.

1 INTRODUCTION

Model-View-Controller (MVC) has been proposed by Trygve Reenskaug in the late seventies as a software design pattern, which aims at modularising the software systems into the model, view, and controller elements and maximising their understandability and analysability (Reenskaug, 2003). Models represent the system data, views represent the pictorial forms on which the users see the model data, and controllers represent the interactions between the views and models (i.e., how the models need to be used and manipulated upon any user requests via the views). Separating the software development into different concerns maximises the cohesion and minimises the coupling, which is one of the main principals of quality coding and aids in reducing the development and maintenance cost. Also, separating the views from the business logic and data enables multiple users (e.g., web designers and developers) to work in parallel, which reduces the development time.

MVC has first been applied in the Smalltalk 80 language in the late eighties, where the model, view, and controller concepts are represented as abstract entities that can be used for creating understandable software systems (Burbeck, 1987). Later on, MVC has gained huge popularity in the area of web applications development, and many web technologies (e.g., .NET, JavaScript, and Php) nowadays offer their own frameworks for the MVC-based web developments. Moreover, as discussed in Section 6, many researches have been conducted on extending MVC for various domain problems (e.g., secure systems, multi-agent systems, and service-oriented systems).

However, the current MVC frameworks force practitioners to work on the software implementation and deal with the coding of the view, model, and controller units. While the MVC frameworks provide many re-usabilities, developers are still expected to deal with low-level programming issues that may require expertise on particular software languages and technologies. Therefore, the effort required for developing and testing the model, view, and controller elements separately may be unexpectedly high. Besides, other (probably non-technical) stakeholders (e.g., customers) may not be involved in the development and testing process, which may negatively affect the quality of software systems. To avoid these,

Ozkaya, M. and Fidandan, I.
MVCLang: A Software Modeling Language for the Model-View-Controller Design Pattern.
DOI: 10.5220/0009571400750083
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 75-83
ISBN: 978-989-758-443-5

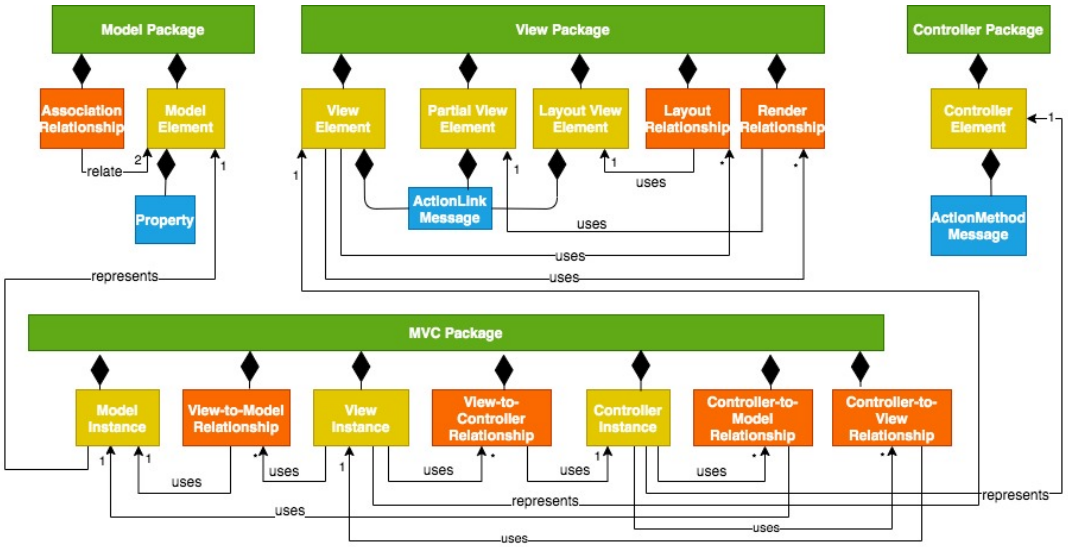

Figure 1: The meta-model of the MVCLang language.

one could enhance the level of abstraction by applying software modeling, which promotes specifying high-level software models that consist of abstract, explanatory statements without any platform-specific implementation details(Seidewitz, 2003). The high-level models are potentially useful for the communication of the design decisions with different stakeholders who can also contribute to the application development and testing despite having very little (or no) experience in programming. The models can be useful for the early detection of the design errors that may otherwise go undetected until the system deployment and cause huge maintenance cost. The models that are analysed to be correct can be used for generating quality code that reflect the modeled design decisions. However, as discussed in Section 6, software modeling has not yet been systematically employed for the development of MVC-based software applications. While many approaches have extended the MVC design pattern for different needs, no any modeling platform consisting of a modeling language (i.e., the notation set) and its toolset has been proposed so far for the MVC-based modeling, analysis, and implementation of software systems.

The goal of this paper is to propose a software modeling language called MVCLang for the MVC-based architecture modeling of software systems. MVCLang offers a visual notation set for the specifications of the model, view, and controller elements of software systems and their relationships. MVCLang is supported with an Eclipse-based prototype toolset. The toolset's modeling editor enables to create the visual MVC architectures and check correctness for a set of wellformedness rules. The toolset's code generator translates MVC models into software code using the ASP.NET MVC framework.

In the rest of this paper, we initially introduce the MVCLang language (i.e., syntax and semantics) and then introduce MVCLang's prototype toolset (i.e., the model editor, analyser, and code generator). Next, we illustrate MVCLang via the sports store case-study and evaluate the language in industry. Lastly, we provide the discussions of the related works.

2 OVERVIEW OF MVCLang

MVCLang offers a visual notation set for the modeling of MVC-based software architectures, which are separated into four packages: model, view, controller, and MVC. While the model, view, and controller packages are created for modeling the model, view, and controller elements of the MVC architectures, the MVC package is created for establishing the relationships between model, view, and controller elements that are modeled in their own packages. Figure 1 shows MVCLang's meta-model, which describes the modeling concepts composing each package and their relationships. The visual symbols that are associated with the modeling concepts are depicted in Figure 2, Figure 3, Figure 4, and Figure 5, which are for the model, view, controller, and MVC packages respectively. In the rest of this section, we discuss for each modeling package the syntax and semantics of the language elements informally. That is, we describe the visual symbols for the language concepts and their relationships, the constituting parts of the language concepts and relationships, and how those concepts

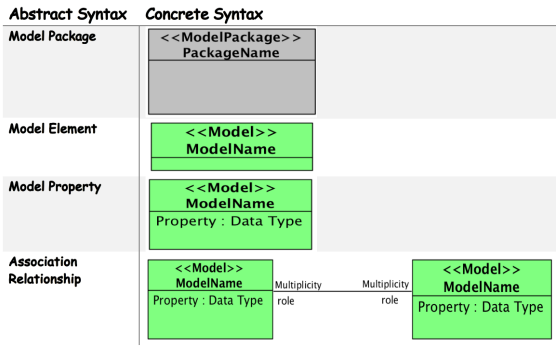

Figure 2: The abstract and concrete syntax for MVCLang's model package.

and relationships can be used and composed together to create meaningful specifications for each package.

2.1 Model Package

As depicted in Figure 1, a model package is specified with a set of model elements that have association relationships between each other. Models represent a dataset to be stored and manipulated by the software system, which may be implemented as database tables. Figure 2 displays the concrete symbols that correspond to the abstract concepts given in Figure 1. So, each model consists of a set of properties that represent the model data and are specified with the data type and its name. The data type can be either *integer*, *boolean*, *string*, or *double*. The association relationship is used to relate the data of different model datasets that are dependent on each other (e.g., the company and employee datasets). The association relationship is specified with the multiplicities and roles, where multiplicities can be one-to-one, one-to-many, many-to-one, and many-to-many and role describes the how each dataset acts in the relationship.

2.2 View Package

Figure 3 shows the symbols for the view package. The partial view represents any re-usable view content(s) that can be used (or rendered) as part of other view(s). The layout view represents the common user-interface parts (e.g., the footer and header), which are shared among different views for offering a consistent layout appearenace. As in Figure 1, each view may have zero or more layout or render relationships, and each relationship enables the view to render/use exactly one partial/layout view. A view of any type is specified with *action-link* or *form* messages that are triggered by the users who interact with the view and cause the view to send requests to the controllers. An action-link message represents

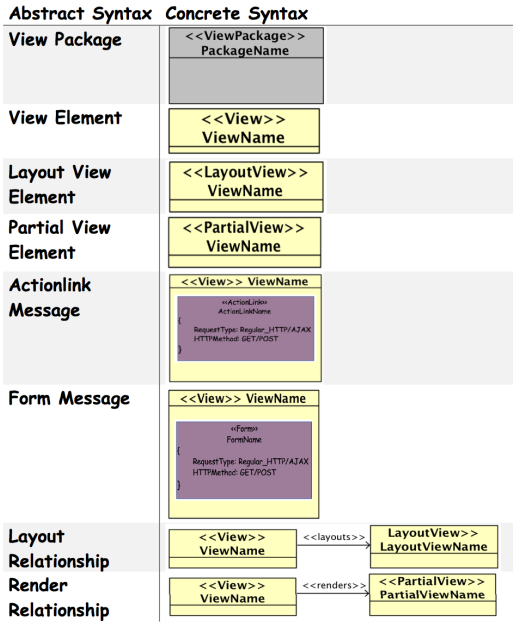

Figure 3: The abstract and concrete syntax for MVCLang's view package.

a link displayed on the view's user interface, which can be clicked by the user. An action-link is specified with the *(i)* name, *(ii)* request type, and *(iii)* http method.The request type can be an HTTP request that prompts the controller to return with a view or an AJAX request that do not prompt for a new view. The http method can be GET or POST. A form message represents the action that results from a form displayed on the view and transmits the data filled in the form to the controller. The form messages are specified with the *(i)* form name, *(ii)* request type, and *(iii)* http method.

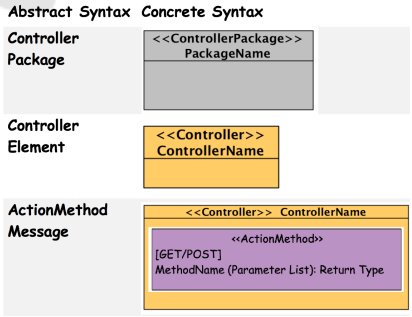

Figure 4: The abstract and concrete syntax for MVCLang's controller package.

2.3 Controller Package

As depicted in Figure 1, a controller package is specified with controller elements that each coordinates the

interactions between the view and model elements. A controller includes a set of *action methods*, which are triggered by the view request (i.e., action-link or form) messages and perform the necessary operations (e.g., adding/removing data, displaying a new webpage, performing some computations, etc.) followed by an output to be returned. Figure 4 shows the symbols for specifying controllers and their action methods. An action-method is specified with *(i)* an http method, *(ii)* its name, *(iii)* its parameter list, and *(iv)* a return type. The http method can be GET or POST. The method name can be any identifiers. The parameter list has zero or more parameters that each describes the data to be passed from the view whose action link (or form) message is connected with the action-method. A parameter is specified as a pair of data type (i.e., integer, string, double) and identifier. The return type is for specifying what output the controller method will return after its computation. The return type can be either *(i)* view, *(ii)* partial view, *(iii)* redirect to action, or *(iv)* content. The view return type is specified when the action-method returns a view to the user, while the partial view return type is when a partial view is returned. The redirect-to-action herein creates a call for another action-method of the controller. The content option is specified when a data of any supported type is returned.

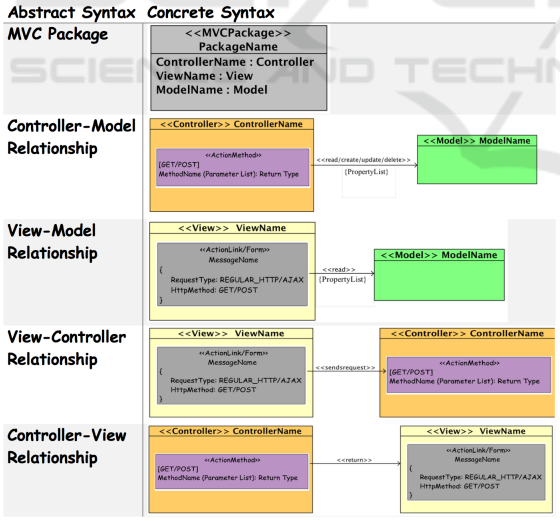

Figure 5: The abstract and concrete syntax for MVCLang's MVC package.

2.4 MVC Package

As Figure 1 shows, an MVC package is specified with *(i)* the instances of the model, view, and controller elements that are specified in the model, view, and controller packages and *(ii)* their relationships. Figure 5 shows the symbols for the MVC package elements and their relationships. Concerning the relationships, each receives a request from one type of element and directs the request to another type of element as specified in Figure 1. The relationship between a view and controller is for connecting the action-link (or form) message of a view with the action-method of a controller so as to enable the view to send requests to the controller. The link arrow herein is by-default specified with the *sendrequests* stereotype. Each action link (or form) message of a view may be linked with exactly one controller method. The relationship between the view and model is for connecting the view with any model whose data are to be displayed on the view. The link's stereotype is *read*, and users can attach the list of properties to be displayed. A view may be connected with zero or more models. The relationship between controller and model is for connecting controller's action-method with the model element so as to perform read/create/update/delete operations on the model data. Users may attach the proper stereotype to the link that indicates the operation and the list of model properties on which the operation is performed. The stereotype can be either *read, create, update,* or *remove*. The operation *read* is for reading the model data specified in the property list, *create* is for creating a new model data, and *update* is for updating the model data given in the property list. The property list may be left empty for creating a model data. Each controller method may be connected with zero or more models. Lastly, the relationship between controller and view is for connecting the action-method with a view element to indicate what view the method returns after its computation. An action-method of a controller may return zero or one view elements.

3 MVCLang's TOOL SUPPORT

We developed a prototype tool for MVCLang using the Sirius meta-modeling tool (Viyović et al., 2014)[1]. Sirius is essentially an Eclipse project that uses and extends the Eclipse Modeling Framework (EMF) for the meta-modeling activities. With Sirius, we created Ecore meta-models to define our language syntax and semantics in EMF and automatically obtained a modeling editor that can be used in the Eclipse platform. Also, we used Eclipse's Acceleo code generation technology[2], which enabled us to integrate a code generator to the modeling editor that we obtained in Sirius. Our Eclipse-based prototype toolset (i.e.,

[1]Sirius: www.eclipse.org/sirius/
[2]Eclipse Acceleo: www.eclipse.org/acceleo/

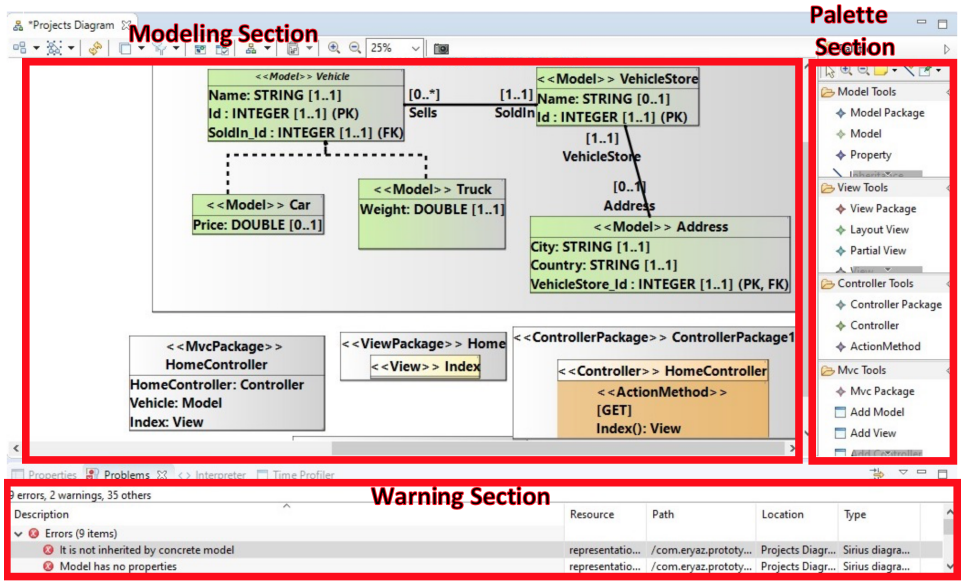

Figure 6: The snapshot of the modeling editor for MVCLang.

Table 1: The well-formedness rules for MVCLang.

Modeling Packages	Wellformedness Rules	
Model	Model element name must be unique in a model package	Model property name must be unique in a model element
	Each model element must have at least one property	Model association relationships must be acyclic
View	View element name must be unique in a view package	Each layout/partial view must be used by at least one view element
	A message http method must be either GET or POST	A message request type must be either HTTP or Ajax
	Each view message (i.e., either action-link or form) must be specified with a name, request type, and http method	
Controller	Controller element name must be unique in a controller package	Each controller element must have at least one action-method message
	Each controller message must be specified with a name, http method, and return type	A message http method must be either GET or POST
	A message return type must be either View, Partial-View, Redirect-to-Action, or Content	
MVC	Each view/model/controller element must be used in at least one MVC package	
	Each action-link/form message of each view element must send a request to exactly one action-method of any controller element	
	Each action-method message of each controller element must receive a request from at least one action-link/form message of any view element	
	If the request type of a view message is HTTP, the controller which receives the request needs to return a view page	
	Any controller element connected with a model element must specify at least one model property that it needs to access	
	The http method of any controller's action method must be the same as that of the view message from which the action method receives a request	

modeling editor and code generator) can be downloaded via the project web-site[3]. As depicted in Figure 6, the modeling editor consists of three sections: modeling section, palette section and warning section. The warning section displays any warning messages due to the violation of the language's wellformedness rules that are presented in Table 1. Whenever an architecture model violates any of those rules, the editor warns the user with an appropriate message.

Our code generator produces ASP.NET MVC framework code[4] from the MVC models specified in MVCLang via our Eclipse modeling editor. In our code generator development, we firstly defined the translation algorithms for producing the ASP.NET MVC code informally and tested their correctness via some simple case-studies. Note that due to the space limitation, we are not able to discuss the translation algorithms here. Then, we used Acceleo to develop the generator that generates ASP.NET MVC code in accordance with the translation algorithms. The generated ASP.NET MVC code essentially consists of three folders, i.e., view, controller, and model. The view folder includes an *.cshtml* file for each view element. The controller folder includes a *C#* file for

[3]MVCLang: sites.google.com/view/mvclang
[4]Asp.net:dotnet.microsoft.com/apps/aspnet/mvc

each controller element, where a *C#* class is generated that includes a distinct method for each action method of the controller. The method body herein includes the relevant algorithm implementation for, e.g., manipulating the model data and displaying data on a view's web user-interface, which are generated from the MVC relationships between the controller and model, view elements. The model folder includes a program file in *C#* for each model element, which is responsible for connecting to a database in an MS SQL Server and creating the necessary table that includes a distinct column for each model property. The resulting code folders can be imported in any Visual Studio .NET environment so as to use the generated ASP.NET MVC framework code.

4 SPORTS STORE CASE STUDY

To illustrate MVCLang, we considered the sports store application (Freeman, 2013). The sport store offers a product catalog for the customers to browse by category and add/remove any product to their shopping cart. Customers can checkout after entering their shipping details. The administrators may also monitor sports store application and change the product catalogue by adding/removing/editing products.

In the rest of the section, we present the MVC package models for the sports store application. Note that due to the space limit, we give here the partial specification that does not include the view, model, and controller package specifications. The whole specifications are available via the project website[3].

The product MVC package is depicted in Figure 7. Whenever the *CartHomePage* view sends a request to the *ProductController* via its *Continue Shopping* action-link message (i.e., the user clicks to continue shopping), the controller receives the request with its *GetProductCategoryList* action-method and returns the *ProductCategoryListPage* view. The *ProductCategoryListPage* view reads the product category data from the *Product* model and displays them. Also, *ProductCategoryListPage* view includes the *ProductListPage* view as a partial view. So, when the user selects a product category via the *ProductCategoryListPage* view, the view's *ChooseProductCategory* action-link message sends the selected category to *ProductController*'s *GetProductList* action-method and that returns the *ProductListPage* view which reads and displays the *Product* model dataset that matches with the chosen product category.

The MVC package for the shopping cart is depicted in Figure 8. The *CartOperationsPage* view that is the partial view for the *ProductListPage* receives add and remove requests from the user and direct them to the controller via its *Add to Cart* and *Remove Product from Cart* action-link messages respectively. Whenever, *CartController*'s action-methods receive a request, the action-methods return the corresponding views that include the form for performing the add/remove operations. That is, the *addToCart* action-method returns the *AddtoCartPage* while the *RemoveFromCart* action-method returns *RemoveFromCartPage*. When the users fill in the forms on the corresponding views, the views send form requests to the CartController's action-methods with the appropriate parameter arguments. The arguments herein are the product ID and amount of products added/removed. CartController's action methods modify the *CartLine* model accordingly. Note also that the action-links and action-methods for the form requests are specified with the *POST* http method.

The administrator MVC model is depicted in Figure 9. Whenever the user clicks to add, edit, or delete products, the *AdminHomePage* view's *Create a new Product*, *Edit a Product*, and *Delete a Product* action-link messages are triggered respectively. Then, the action link messages forward the requests to the connected action methods of *AdminController*. The *AdminController* responds with the appropriate form views (e.g., if delete request is received, the *DeleteProductPage* form view is returned). When the user submits the form, the corresponding view's form message is sent to *AdminController*'s connected action-method and the action method accesses the *Product* model data so as to make the necessary changes. Note that the action-link and action-method messages for handling the form requests are specified with the *POST* http method.

We used our modeling editor for analysing Sport store's MVC architectures for the wellformedness rules given in Table 1. Given all the rules being satisfied, we used the code generator to produce the ASP.NET MVC framework code. Figure 10 shows the user-interfaces that have been generated from the sport store's view specifications. The full generated code can be accessible via the project web-site[3].

5 EVALUATION

We evaluated MVCLang and its toolset in a software company called Eryaz Software[5], which have offered MVC-based e-commerce systems to hundreds of different customers since the nineties. In Eryaz, we asked 5 experienced developers to use MVCLang for

[5]Eryaz Software: www.eryazsoftware.com.tr/en

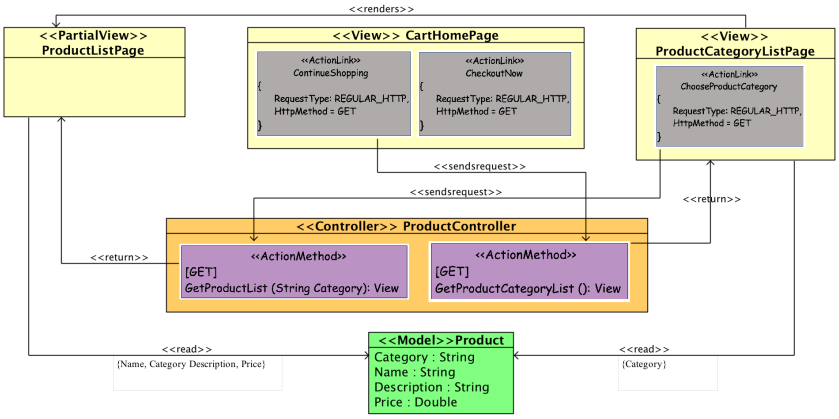

Figure 7: The MVC package model for the product controller in sports store.

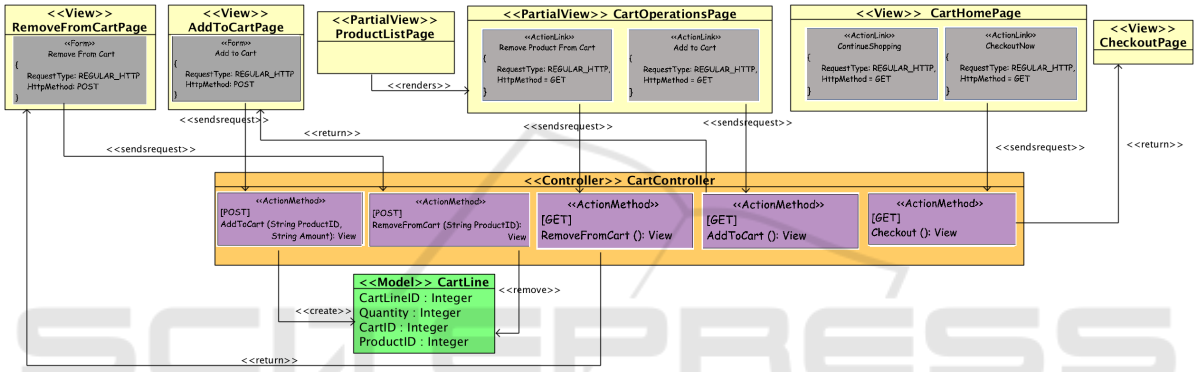

Figure 8: The MVC package model for the shopping cart controller in sports store.

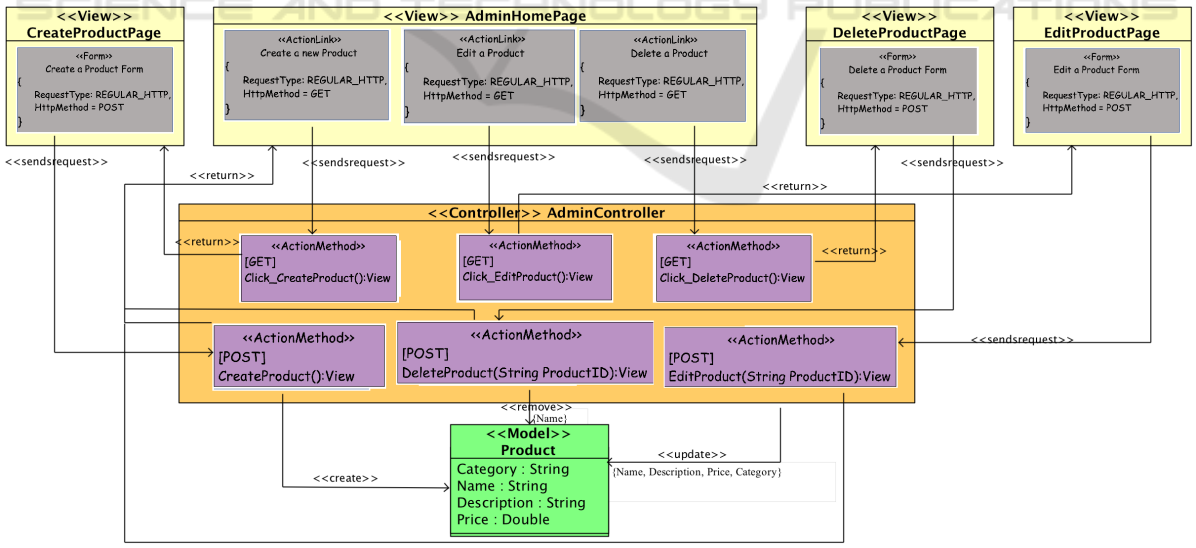

Figure 9: The MVC package model for the admin controller in sports store.

any project that they develop and share their feedback with us. We separated a 2-hour slot for each developer. During each session, we observed the developer's modeling activities and answer the questions that the developer asked. After completing the session, we requested each developer to answer the following questions:

1. Do you find the notation set for the MVCLang ad-

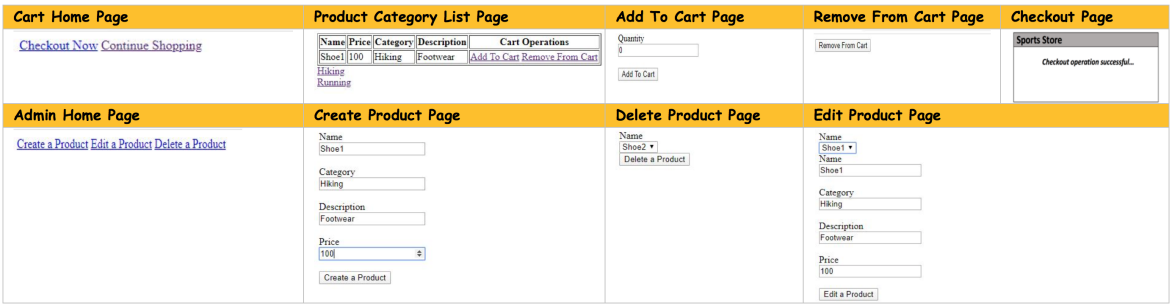

Figure 10: The web-pages generated from the sports-store's architectural models in MVCLang.

equate for MVC-based architecture modeling?

2. Is MVCLang easy to learn and use?
3. Does MVCLang shorten the development time for the MVC-based web applications?
4. Is the generated code from MVC architectures useful?

So, we observed interesting findings. Concerning the first question, the developers are satisfied with the language's notation set. However, one developer asked if we could enable modeling the views in terms of user-interface elements such as labels, text-boxes, and buttons and their layout. Also, two developers requested a user-manual that describes the language syntax. Another developer asked if we plan to make MVCLang extensible to enable users to introduce domain-specific notations for the MVC-based modeling. Concerning the second question, the developers found MVCLang easy to learn and use as MVCLang resembles the simple boxes-and-lines and UML that many practitioners are familiar with. The developers were however concerned about MVCLang's support for large and complex MVC architectures, as MVCLang does not currently support any techniques for managing large models (e.g., sub-diagramming and composite architectures). Concerning the third question, some developers agreed that using MVCLang for the automated generation of ASP.NET code from the MVCLang specifications aids in shortening the development time and maximising the software quality as the code is expected to meet the design decisions. However, some code-oriented developers pointed out the time needed for generating correct architectural models that may sometimes outweigh the time needed for just-coding. Lastly, the developers are quite happy with the generated code that can be used with some little changes to reach an executable web application. One of the developers asked whether we plan to provide a reverse-engineering support so as to get back to an MVCLang specification after the generated code is changed.

6 RELATED WORK

Many software modeling languages, including UML (Rumbaugh et al., 2004) and its derivatives, architectural languages, and domain-specific modeling languages, are available for creating software models from different perspectives at different levels of abstractions and performing useful operations such as model analysis, formal verification, and code generation. However, our recent analyses on the existing languages (Ozkaya, 2018a; Ozkaya, 2018b) let us observe that none of those languages provide a modeling notation set for the MVC-based modeling and rather focus on particular domains (e.g., embedded systems, multi-agent systems, and distributed systems) or provide general-purpose notation sets. The only exception here is UWE, which extends UML for the web-applications modeling. UWE (Kraus et al., 2007) does not directly adopt the MVC pattern and rather supports the separation of concerns in terms of the content, navigation, presentation, and process viewpoints. While the presentation viewpoint corresponds to the view element in MVC, the model and controller elements of MVC are not explicitly considered in UWE. Also, it is not so easy to model the relationships between different viewpoint models that are each specified as a separate UML diagram. Unlike our toolset that generates code in ASP.NET, UWE's toolset generates code in JSF (Java Server Faces) framework.

The literature also includes some works that attempt at extending the MVC pattern for different needs. In (Taraghi and Ebner, 2010), Taraghi et al. extended MVC for designing and developing web-based widgets that are to be used for the personal learning environment implementation at TU Graz. In (Delessy-Gassant and Fernandez, 2012), Delessy-Gassant et al. considered extending MVC with the modeling of some security requirements such as authentication, role-based access control, secure logging, and secure data traffic. In (Sauter et al., 2005),

Sauter et al. proposed another extension on MVC for the development of pervasive multi-device web applications. In (Barrett and Delany, 2004), Barrett et al. extended MVC with a 5-tier architecture. The first tier is the client, the second tier is the view, the third tier is the business logic, the fourth tier is the data abstraction, and the fifth tier is the persistent data in the database. In (Mahmoud and Maamar, 2006), Mahmoud et al. proposed an approach for applying MVC for the modeling of multi-agent systems. In (Cortez and Vazhenin, 2015), Cortez et al. extended MVC for the design and development of service-oriented architectures. While MVC have been extended many times for different problem domains, none of those works actually offer a modeling notation set for the MVC-based modeling and a supporting toolset for the modeling, analysis, and code generation.

7 CONCLUSION

In this paper, we proposed a software modeling language called MVCLang for the MVC-based modeling of software architectures. MVCLang provides a visual notation set for specifying the model, view, and controller elements, and their relationships. We also developed a modeling editor using the Eclipse Sirius technology, through which practitioners can create their MVC-based software models and check their models for a number of wellformedness rules that we defined. Furthermore, we also developed a code generator that can be used together with a modeling editor for producing ASP.NET MVC framework code from the MVCLang models. To illustrate the use of MVCLang, we considered in the paper the sports store case-study, which we specified, analysed, and implemented using MVCLang and its toolset. To evaluate MVCLang, we considered a software company called Eryaz, which builds MVC-based e-commerce solutions for their customers. We prompted 5 developers from Eryaz to use the MVCLang language and its toolset for their projects and share their feedback with us. We also asked the developers a pre-determined set of questions to learn their thoughts.

We are currently in the process of modifying the language definition and the prototype toolset in accordance with the feedback received from the developers and releasing a newer version of the language. We also plan to design some case-studies from diverse industries for evaluating MVCLang so as to determine whether *(i)* the language definition (i.e., syntax and semantics) is adequate for modeling diverse problems and *(ii)* the toolset detects the design errors and produces the expected ASP.NET code correctly or not.

REFERENCES

Barrett, R. and Delany, S. J. (2004). openmvc: A non-proprietary component-based framework for web applications. In *PROC. 13 TH INTERNATIONAL WORLD WIDE WEB CONFERENCE*, pages 464–465.

Burbeck, S. (1987). Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc).

Cortez, R. and Vazhenin, A. (2015). Virtual model-view-controller design pattern: Extended mvc for service-oriented architecture. *IEEJ Transactions on Electrical and Electronic Engineering*, 10(4):411–422.

Delessy-Gassant, N. and Fernandez, E. B. (2012). The secure mvc pattern. In *1st LACCEI International Symposium on Software Architecture and Patterns*, pages 1–6.

Freeman, A. (2013). *Pro ASP.NET MVC 5*. Apress, USA, 5th edition.

Kraus, A., Knapp, A., and Koch, N. (2007). Model-driven generation of web applications in UWE. In Koch, N., Vallecillo, A., and Houben, G., editors, *Proceedings of the 3rd International Workshop on Model-Driven Web Engineering MDWE 2007, Como, Italy, July 17, 2007*, volume 261 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Mahmoud, Q. H. and Maamar, Z. (2006). Applying the mvc design pattern to multi-agent systems. In *2006 Canadian Conference on Electrical and Computer Engineering*, pages 2420–2423.

Ozkaya, M. (2018a). Analysing uml-based software modelling languages. *Journal of Aeronautics and Space Technologies*, 11(2):119–134.

Ozkaya, M. (2018b). The analysis of architectural languages for the needs of practitioners. *Softw., Pract. Exper.*, 48(5):985–1018.

Reenskaug, T. (2003). The model-view-controller (mvc) its past and present.

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education.

Sauter, P., Vögler, G., Specht, G., and Flor, T. (2005). A model-view-controller extension for pervasive multi-client user interfaces. *Personal Ubiquitous Comput.*, 9(2):100?107.

Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

Taraghi, B. and Ebner, M. (2010). A simple mvc framework for widget development. In *Proceedings of the 3rd Workshop on Mashup Personal Learning Environments (MUPPLE10)*, pages 1–8. . ISSN 1613-0073.

Viyović, V., Maksimović, M., and Perisić, B. (2014). Sirius: A rapid development of dsm graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238.