A Model-Driven Development of Web Applications Using AngularJS Framework

Wutthichai Chansuwath
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
wutthichai.c@student.chula.ac.th

Twittie Senivongse
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
twittie.s@chula.ac.th

***Abstract*—AngularJS is one of the widely used frameworks for modern single-page web application development which is designed to support dynamic views in the applications. To further assist AngularJS developers, this research proposes how the concept of model-driven development can be applied to AngularJS-based development. We propose a UML profile for AngularJS for building a model of an AngularJS web application, and a set of transformations that transform the model into a code template. The developer can then fill in the template to make a complete workable web application. Also, a transformation tool is developed to assist in constructing the code template. Using a case study application, the evaluation in terms of transformation rate shows that the automatically generated code covers 87% of the complete code of the case study, which means it could greatly help reduce development time.**

Keywords—model-driven development; AngularJS; UML profile; web applications

I. INTRODUCTION

Software companies have the needs to deliver software products as fast as possible to stay competitive, and have to find the way to reduce time to market while producing quality software for customers. One response to such needs is the concept of model-driven development that uses software models to drive software development. OMG's Model Driven Architecture (MDA) [1] promotes model-driven development through transformation of one software model into another model. Software models can be at different levels, e.g. Platform-Independent Model (PIM) level that is a software model that represents business functions, Platform-Specific Model (PSM) level that is a software model that focuses on business functions implemented on a specific platform, and code level. A model at one level can be automatically generated from a model at another level based on a set of transformations. This mechanism does not only help reduce time to develop but also lower the risk of introducing bugs during development and promote application portability.

The model-driven development concept has been applied to development of various kinds of software, including web applications. Web application technology has improved significantly over time in such a way that a web application is now much more like a desktop application, e.g. in terms of performance. One technique to improve web application performance in terms of response time is by creating the web site as a single page application [2]. In a single page application, all code is retrieved with a single load of the page or resources dynamically loaded to the page, without constant reloading of the page or transferring of control to other pages. Almost everything is done on the client side of the application. One of the frameworks for building a web site as a single page application is Google's AngularJS [3][4]. Due to its popularity, we see that applying the model-driven development concept to AngularJS-based web applications would largely benefit web application development.

In this research, the development of web applications will be driven by the models at the PSM level, i.e. the models for the AngularJS platform. We use the UML profile mechanism [5] to extend UML for modeling AngularJS-based applications. A web developer can use this UML profile for AngularJS to build the design model of a web application. Based on the proposed transformation templates, the developer can fill in the code template that is generated from the model in order to build a complete application. We also develop a transformation tool and report an evaluation of the approach, using a financial application developed by a large software company.

Section II of the paper presents some background and related work. Section III describes the UML profile for AngularJS and transformations for code generation which are applied to a case study in Section IV. An evaluation is reported in Section V and the paper concludes in Section VI.

II. BACKGROUND AND RELATED WORK

A. Model-Driven Development and UML Profile

As mentioned previously, the model-driven development concept, like MDA [1], can be used to generate software code from the models. The development team can respond to business requirements by first building a software model, e.g. in UML, at the functional PIM level. This model can be transformed into lower level models, i.e. the PSM level and code level respectively, by using transformation tools that recognize transformation rules. These rules define mapping between the metamodels that are used to build the models at the two consecutive levels. In this research, the UML profile [5] is used to extend the UML metamodel to create a platform specific metamodel for building PSM-level models. A UML profile allows a definition of stereotypes (as specific metaclasses denoted by << >>), tagged values (as specific

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE
ICIS 2016, June 26-29, 2016, Okayama, Japan

metaattributes), and constraints, which altogether are used to model software on a specific platform, i.e. AngularJS platform.

B. AngularJS

AngularJS [3][4] is a popular JavaScript framework for creating front-end single page web applications. It is designed to support dynamic views which makes browsing the page smooth like that in a native application. Important features are:

1) Model-View-Controller: This architecture separates the application into three layers: (1) *view* corresponds to the user interface shown on a browser, (2) *model* corresponds to the data shown to the users on views, and (3) *controller* corresponds to the logic to control the data shown on views.

2) Template: This is a group of HTML that is converted to Document Object Model (DOM) for showing a user interface.

3) Two-way data binding: This is a mechanism for the view to change with respect to the change of the model.

4) Dependency injection: This is a mechanism to let AngularJS load all services completely before processing anything.

5) Directive: This is a group of the templates that function as programmed by the developers.

C. Related Work

Several approaches to UML-based modeling and model-driven development for web applications exist. For example, Huang et al. [6] propose a UML profile for modeling web applications at the PSM level and have Servlets, JSP, and Java code templates generated. Hsu [7] presents a UML profile for web 2.0 mashups which is used in the model-driven development of a web mashup that uses map and web feed services. Kaewkao and Senivongse [8] base the generation of their Google App Engine applications on the proposed UML profile. Other researchers such as Mubin and Jantan [9] only present a UML profile for modeling conceptual, navigational, and user interface features of web applications, while Kataria et al. [10] present a component-centric UML profile for modeling web applications that use ASP, JSP, PHP, Servlets, and JavaBeans technologies. Similar to these researches, we drive the development of web applications by UML profile-based modeling but target the AngularJS platform.

III. AngularJS-Based Model-Driven Development

This section presents the overview of our approach, UML profile for AngularJS, model-to-code transformation.

A. Overview

Fig 1. illustrates the overview of the approach. Given the software requirements, a system analyst uses a UML modeling tool to build a model for an application based on the UML profile for AngularJS (step 1). The model is exported to the XMI format [11] (step 2) and is then used as an input to the M2C (Model-to-Code) transcoder which is the transformation tool that recognizes transformation templates (step 3). The transformation result is the AngularJS source code template (step 4). Given the code template, a developer will further fill in the business logic to make a complete web application.

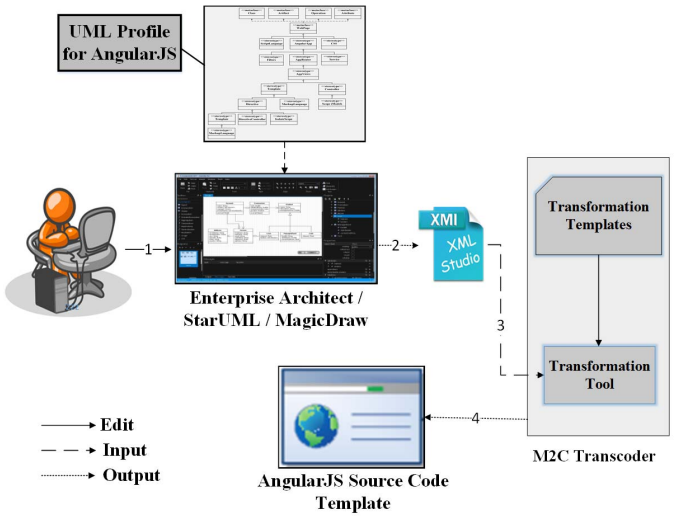

Fig. 1. Overview of the approach.

B. UML Profile for AngularJS

The UML profile for AngularJS is depicted in Fig. 2 and the definition of all stereotypes in the AngularJS domain is listed in Table I.

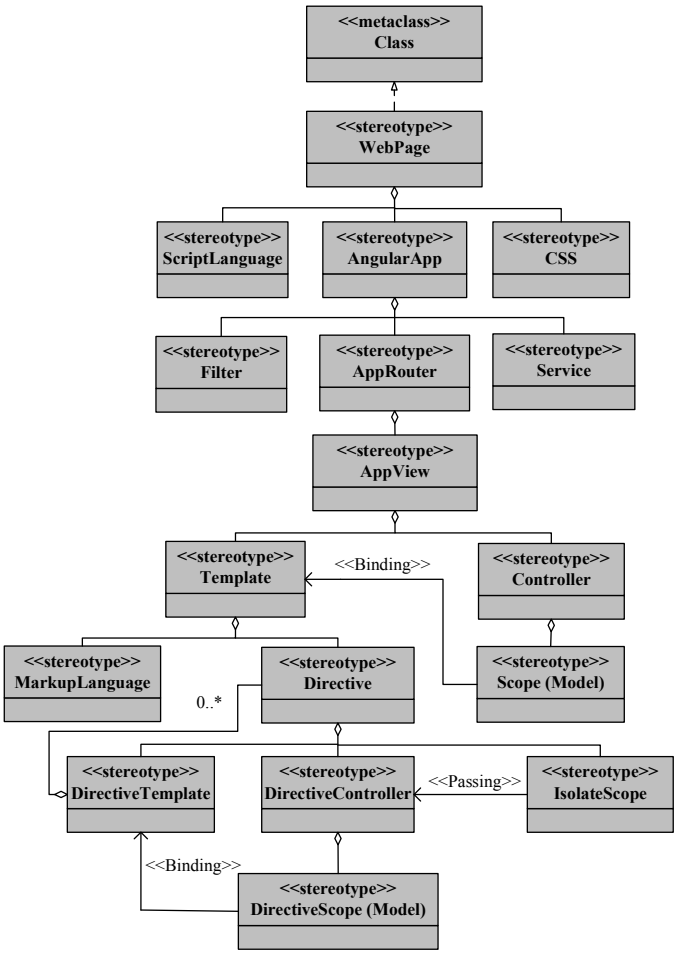

Fig. 2. Overview of UML profile for AngularJS.

TABLE I. STEREOTYPES FOR ANGULARJS

Stereotype	UML Metaclass	Description
AngularApp	Class	Define the whole AngularJS application
AppRouter	Class	Define which view to be navigated to
AppView	Class	Define the whole user interface of the single web page
Binding	Association	Represent the relationship between Scope and Template. If Scope is changed, it will immediately affect Template that is showing the Scope value on the web browser.
Controller	Artifact	Function to process all business logic
CSS	Artifact	CSS tag on the web application
Directive	Artifact	AngularJS directive
DirectiveTemplate	Artifact	Partial HTML of Directive
DirectiveController	Artifact	Function to process all business logic inside a directive
DirectiveScope	Artifact	Model to be used on Directive
Filter	Artifact	Function to format any value of AngularJS application
IsolateScope	Artifact	Model to be used inside directive
MarkupLanguage	Class	HTML tag to render on the client side
Passing	Association	Association between IsolateScope and DirectiveController. IsolateScope will pass the value that is set from the Directive to DirectiveController to process any business logic.
Scope	Artifact	Model to be used on the template
ScriptLanguage	Class	JavaScript file to be used in the web application
Service	Artifact	Shared functions to process specific business logic across AngularJS application
Template	Artifact	User interface of AngularJS view
WebPage	Class	Web page that is sent from web server

In this profile, the AngularApp stereotype denotes an AngularJS web application. It can load a new view that a user wants to see inside the AngularJS web application on the client side by using AppRouter. AppRouter will control how to redirect or navigate to other views. The view is modeled by the AppView stereotype. If the user wants to navigate or redirect to the new view, AngularJS will load that view inside the AngularJS web application without making a request to the server to get the whole HTML to render on the web browser. This mechanism will provide a smooth responsive user experience similar to a desktop application. AppView appears like a normal HTML tag but it can be divided into smaller components called Directives, each for a specific purpose.

Directive uses the concept of the Model-View-Controller architecture and comprises three parts. The first one is DirectiveTemplate (View) which is the HTML to be shown to the user. The second one is DirectiveScope (Model) which is a model to bind with DirectiveTemplate (View), i.e. if the Model is changed, the View is changed accordingly. The last one is DirectiveController (Controller) which is a controller of AngularJS to perform any business logic and set the value back to DirectiveScope for showing on the screen. IsolateScope is a property with a value that is assigned from outside the Directive. The IsolateScope value will be passed to DirectiveController for processing of certain business logic.

An important part of the AngularJS framework concerns how to change the format of data before rendering to the screen. This is referred to as Filter, and the AngularJS profile provides the Filter stereotype for this purpose.

C. *Model-to-Code Transformation*

A number of transformations are defined to map the AngularJS metamodel (in Table I) to code. See Figs. 6 and 7 in the next section. For simplicity, we define these transformations in the form of AngularJS code templates, with placeholders for the corresponding tagged values (i.e. values of stereotype properties) in the model of the application which is based on the UML profile for AngularJS. These templates will be addressed again when we apply them to a case study.

IV. CASE STUDY

We demonstrate how the UML profile for AngularJS and the transformation are applied to a case of a large software company that develops web applications for the financial domain. The case study is a web application that shows a table of stock quotes. Traders or brokers can monitor price movement and compare the prices of different stocks in the table. Fig. 3 shows the screen of this web application. The rest of this section discusses the AngularJS-based model-driven development that results in this screen.

First, a PSM-level model of the application has to be designed based on the metamodel elements in the UML profiles for AngularJS. Fig. 4 shows the overall model. The Directive part (bottom left) of the model is further detailed in Fig. 5. We describe some of the elements (those annotated by a number) as follows.

This single page application is called “App” (1 in Fig. 4). The “App” is an object that the AngularJS framework uses across this web application. After we create an empty App, we have to create a view to show inside the App. We create a view named quotePrice, shown in stateName (2 in Fig. 4), for showing the stock symbols and prices. AngularJS will automatically add a prefix # to the stateUrl (3 in Fig. 4) to navigate to the view, i.e. the URL becomes #/QuotePrice.

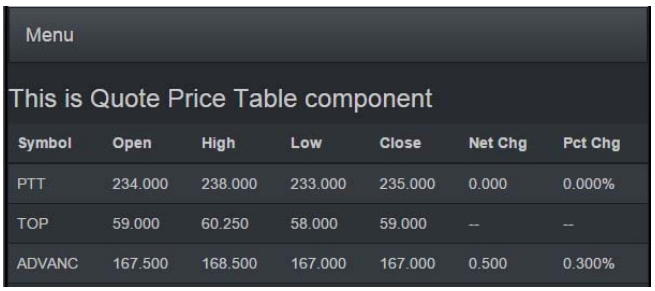

Fig. 3. Screen of case study application.

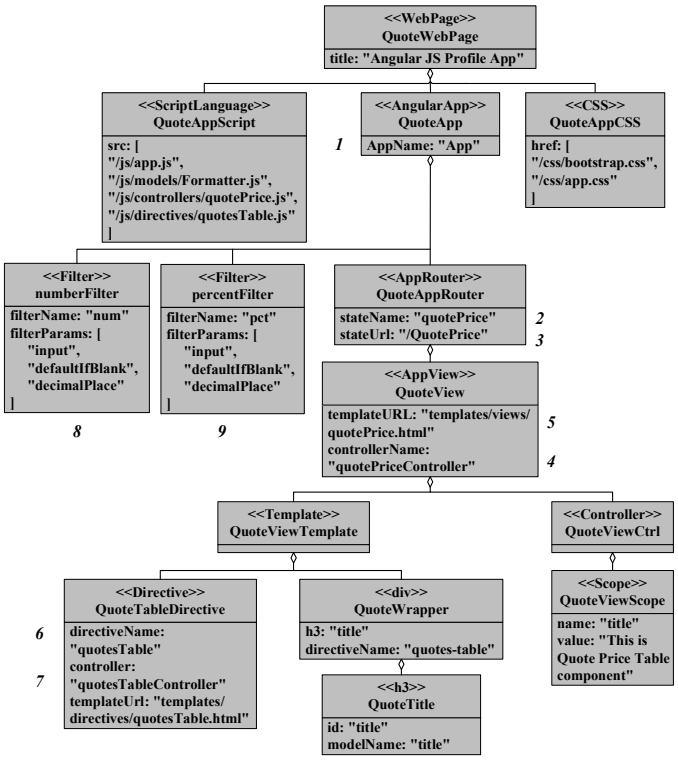

Fig. 4. Design model of case study based on UML profile for AngularJS.

As a server has to host all AngularJS-related files (i.e. AngularJS libraries, HTML, CSS, JavaScript) and respond to a web browser request for the application, the server will generate the prefix URL as http://localhost:9999/App/QuoteApp. Therefore the full URL is http://localhost:9999/App/QuoteApp#/QuotePrice.

The controller of this view is quotePriceController, shown in controllerName (4 in Fig. 4), which performs business logic and has quotePrice.html (5 in Fig. 4) as the view or user interface for the quotePrice view. Besides, the quotePrice view will add a Directive named quotesTable (6 in Fig. 4) for showing the data in a table. The controller of this Directive is quotesTableController (7 in Fig. 4) and the view or user interface of this Directive is quotesTable.html. This directive creates the whole table including the table header (e.g. symbol, open, high, low etc.) for showing the stock symbols and prices. In the quotesTable Directive, we add the AngularJS Filter (1, 2 in Fig. 5) to format the value of each cell before showing on the screen. We also create a Filter called num (8 in Fig. 4) to format the number value to 3 decimal digits. Another Filter is called pct (9 in Fig. 4) which formats the number value to 3 decimal digits and appends a percent sign to the formatted value. In the table in Fig. 3, the num Filter are used with Open, High, Low, Close, and Net Chg columns, whereas the pct Filter is used with the Pct Chg column.

The transformation tool receives the design models in Figs. 4 and 5 (in an XMI format) as input to the transformation. The tool recognizes the transformation templates in Figs. 6 and 7. The key component of each template is the placeholder $<<stereotype>>_{property}$ which will be replaced by the tagged value of the stereotype property of the model element.

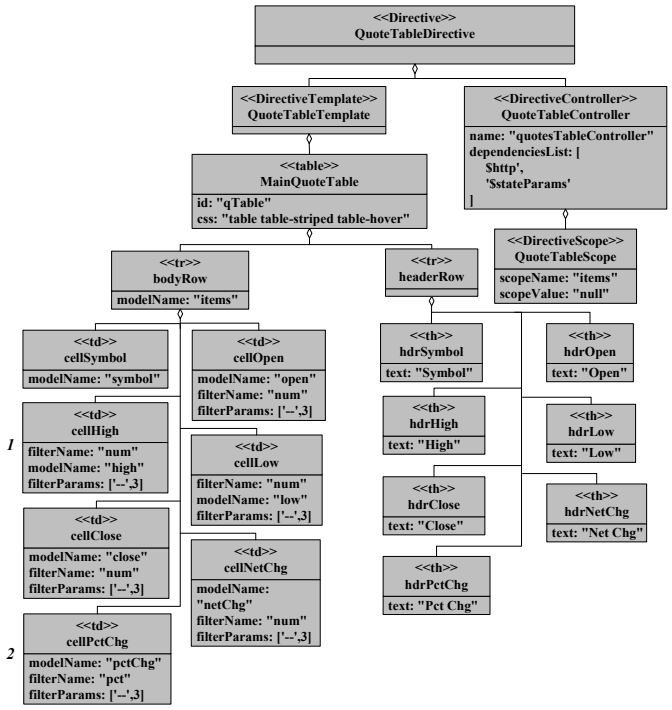

Fig. 5. Directive part of design model of case study.

The rightmost column of Figs. 6 and 7 shows the generated AngularJS source code template for the case study. More business logic has to be added to the code to make a complete stock quote application.

V. EVALUATION

In an evaluation, we determine the performance of the approach in terms of transformation rate [6], i.e. the percentage of code that can be automatically generated compared to the complete code after the developer has filled in the missing business logic for the application. Even though the transformation rate can vary as it depends on the size and complexity of the added logic, it can give an idea of how much effort can be saved in the implementation of a particular application. We measure the code size in terms of the number of lines of code (LOC). The LOC of the code generated for the case study application is 95. We have six developers, each with experience in AngularJS, add necessary code to the generated code template to make a complete application. The results are shown in Table II. The generated code covers between 84-90% of the complete code for the more- and less-experienced groups respectively, i.e. around 87% on average.

TABLE II. EVALUATION RESULT

Experience	Average Manually Added LOC	Average Total LOC	% of Manually Added LOC
3 years or more (3 people)	11	105	10%
Less than 3 years (3 people)	18	113	16%

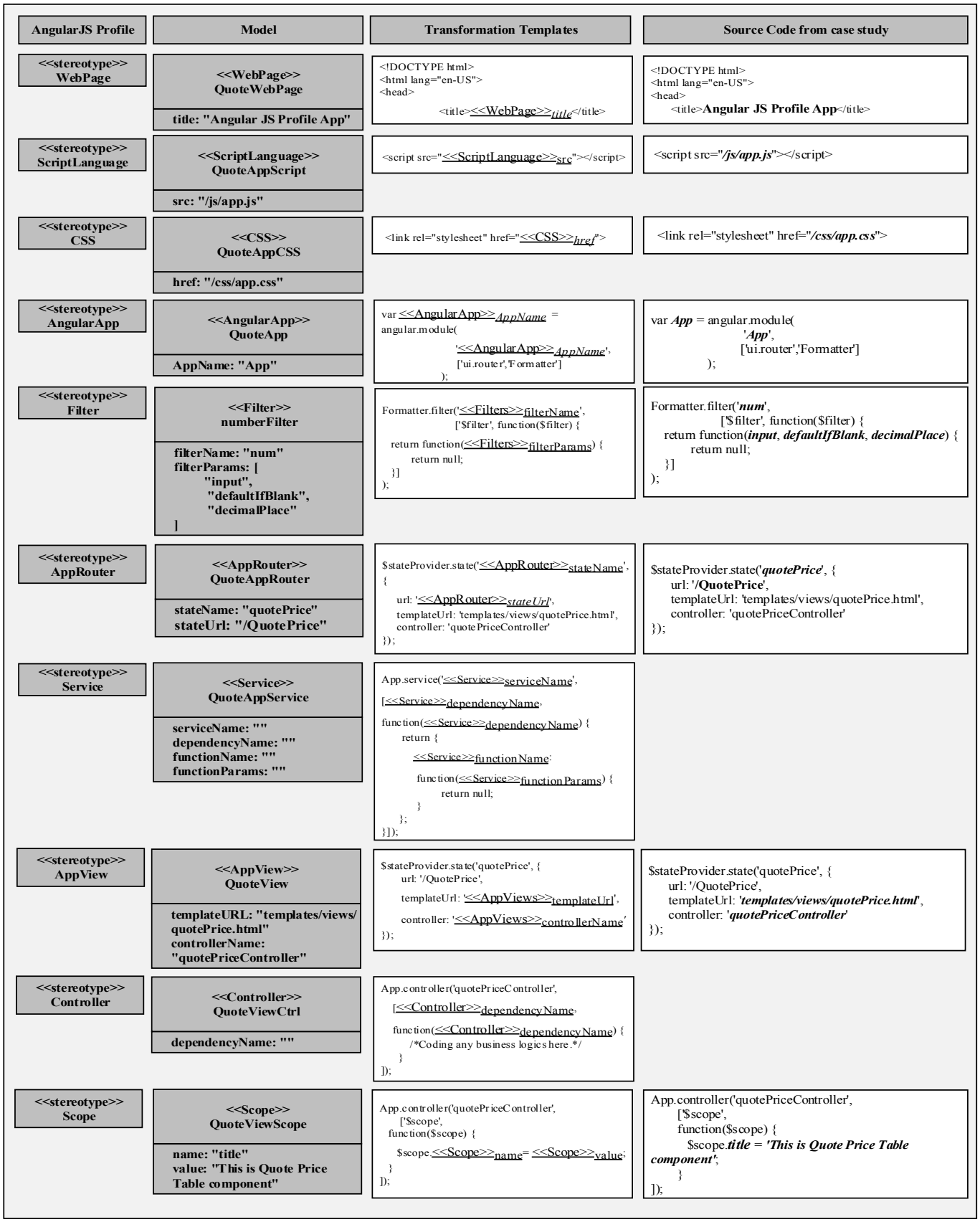

Fig. 6. Transformation templates and example of generated source code for case study (part 1).

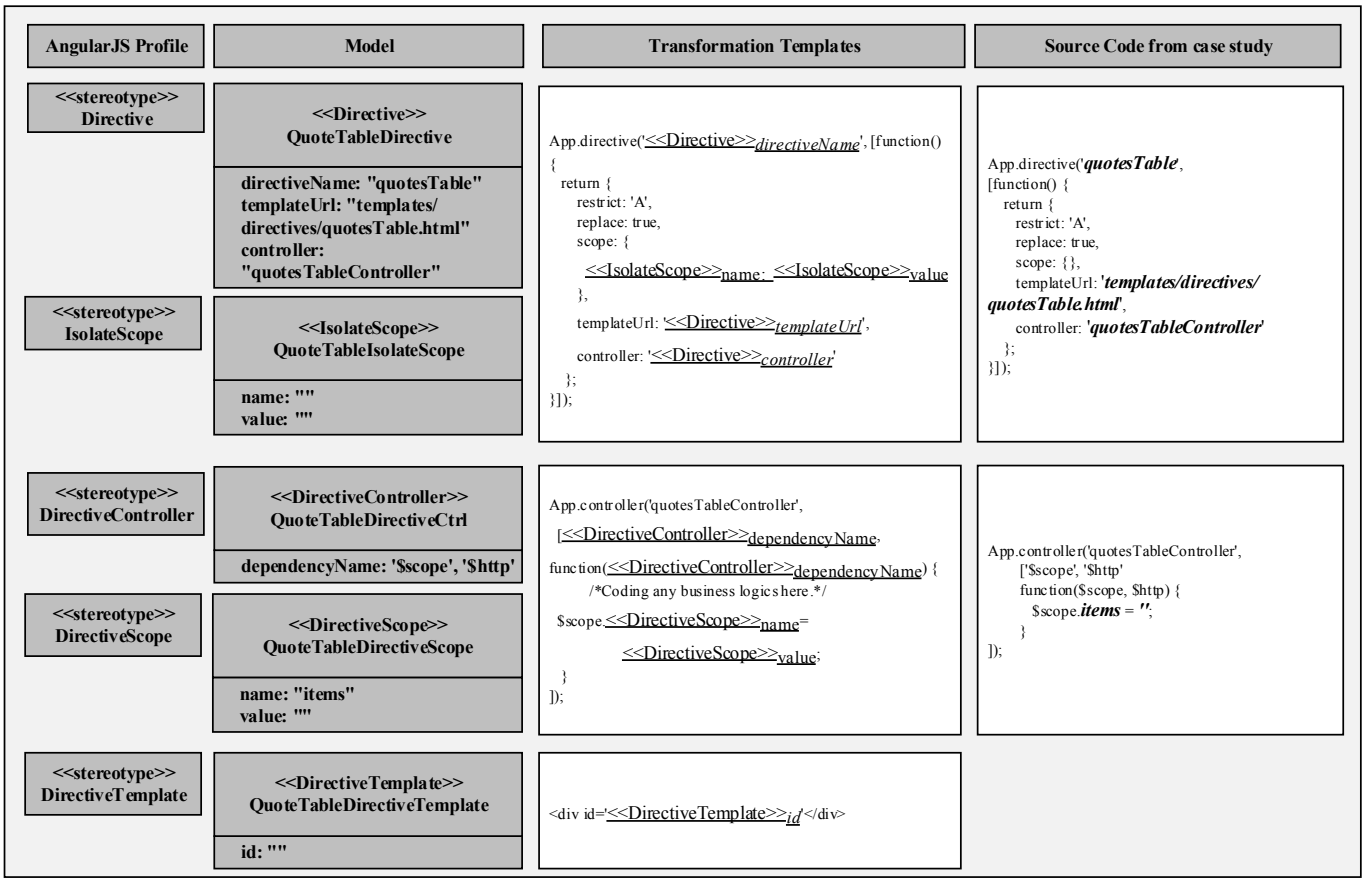

Fig. 7. Transformation templates and example of generated source code for case study (part 2).

VI. CONCLUSION

This paper supports the model-driven development concept and proposes the UML profile for AngularJS and accompanying transformation templates that can be used respectively to model and generate code for AngularJS applications. The evaluation, using a simple case study application, reports that the approach can reduce the number of lines of code that has to be manually incorporated into the application to around 13% on average (i.e. 87% automatically generated). Even though time still has to be spent on the modeling activity, we obtain the models as valuable documentation that can help reduce coding time of the whole web applications. Generally, this can also lower the risk of introducing bugs during development. For future work, a more extensive evaluation is planned for applications of different size and complexity. An extension to the proposed UML profile would be to support modeling of applications in a specific domain. For example, a UML profile for AngularJS for the financial domain should comprise metamodels that correspond to the existing financial metamodel.

REFERENCES

[1] OMG, MDA - The Architecture of Choice for a Changing World [Online]. Available: http://www.omg.org/mda/. Last Accessed: 30 Mar 2016.

[2] M. Mikowski and J. Powell, Single Page Web Applications: JavaScript End-to-End", Manning Publications Co., 2013.

[3] Google, AngularJS. [Online]. Available: https://angularjs.org/. Last Accessed: 30 Mar 2016.

[4] N. Jain, P. Mangal, and D. Mehta, "AngularJS: A modern MVC framework in JavaScript," J. Global Research in Computer Science, vol. 5, no. 12, 2015, pp. 17-23.

[5] OMG, UML Profile. [Online]. Available: http://www.omg.org/mda/specs.htm. Last Accessed: 30 Mar 2016.

[6] Y.-C. Huang, C.-P. Chu, Z.-A. Lin, and M. Matuschek, "Transformation from web PSM to code," Proc. Int. Conf. Distributed Multimedia Systems (DMS), 2009.

[7] I-C. Hsu, "Visual modeling for web 2.0 applications using model driven architecture approach," Simulation Modelling Practice and Theory, vol. 31, Feb 2013, pp. 63-76.

[8] S. Kaewkao and T. Senivongse, "A model-driven development of web-based applications on Google App Engine platform," Proc. 10th National Conf. Computing and Information Technology (NCCIT 2014), Bangkok, Thailand, 8-9 May 2014, pp. 140-145 (in Thai).

[9] S. A. Mubin and A. H. Jantan, "A UML 2.0 profile web design framework for modeling complex web application," Proc. 2014 Int. Conf. Information Technology and Multimedia (ICIMU), 2014, pp. 324-329.

[10] M. Kataria, R. Yadav, and A. Khunteta, "A component-centric UML based approach for modeling the architecture of web applications," Int. J. Recent Research and Review, vol. V, Mar 2013, pp. 22-27.

[11] OMG, Documents Associated With XML Metadata Interchange (XMI), Version 2.4.2 [Online]. Available: http://www.omg.org/spec/XMI/2.4.2/. Last Accessed: 30 Mar 2016.