Virtual Worlds on Demand? Model-Driven Development of JavaScript-based Virtual World UI Components for Mobile Apps

Matthias Stürner and Philipp Brune
Department of Information Management, University of Applied Sciences Neu-Ulm, D-89231 Neu-Ulm, Germany

Keywords: Model-Driven Software Development, Mobile UI, JavaScript, UML Modeling, Virtual Worlds, Computer Games.

Abstract: Virtual worlds and avatar-based interactive computer games are a hype among consumers and researchers for many years now. In recent years, such games on mobile devices also became increasingly important. However, most virtual worlds require the use of proprietary clients and authoring environments and lack portability, which limits their usefulness for targeting wider audiences like e.g. in consumer marketing or sales. Using mobile devices and client-side web technologies like i.e. JavaScript in combination with a more automatic generation of customer-specific virtual worlds could help to overcome these limitations. Here, model-driven software development (MDD) provides a promising approach for automating the creation of user interface (UI) components for games on mobile devices. Therefore, in this paper an approach is proposed for the model-driven generation of UI components for virtual worlds using JavaScript and the upcoming Famo.us framework. The feasibilty of the approach is evaluated by implementing a proof-of-concept scenario.

1 INTRODUCTION

Social virtual worlds and computer games in which the player has to move an avatar character through a 3D oder semi-3D scenery form an important part of the computer games domain and provide also an interesting concept, i.e. for online markerting, sales or product presentation applications. However, existing social virtual worlds and game environments mostly require proprietary client software and lack general acceptance outside their respective user communities, which limits the possibilities for commercial applications (Stangl et al., 2012). Using purpose-specific virtual worlds based on open client technologies embedded into company web sites or mobile apps publicly available would be an alternative.

In recent years, web technologies like JavaScript (JS), HTML5 and CSS3 turned web browsers into universal runtime environments for creating portable user interfaces (UI) of so-called Rich Internet Applications (RIA) or hybrid mobile or desktop apps. These applications offer complex functionality, high interactivity and appealing visual design. Thus, they also form a promising basis for implementing the described purpose-specific virtual worlds. However, to be feasible in practice this would require an efficient and highly automatic implementation of the virtual world software components. Using model-driven software development (MDD) to generate the necessary interactive JavaScript UI components from Unified Modeling Language (UML) models describing the corresponding game scenario on a logical level is one possibility to address this challenge (Huber and Brune, 2012). Therefore, in the present paper an approach is presented for generating UI components from a UML model representing a scenario in which the user navigates a character through a virtual setting. The model is created within the Eclipse IDE using the Ecore framework. From this model a code generator creates the JavaScript code of the UI components using the upcoming Famo.us framework[1]. The feasibility of the approach is evaluated by implementing a proof-of-concept scenario.

The rest of this paper is organized as follows: In section 2 the related work is analyzed in detail. Section 3 describes the design of the proposed UML model and section 4 the implementation of the code generator and its JavaScript library. The proof-of-concept implementation and the evalutaion results are presented in section 5. We conclude with a summary of our findings.

[1]http://famous.org

Stürner, M. and Brune, P.
Virtual Worlds on Demand? Model-Driven Development of JavaScript-based Virtual World UI Components for Mobile Apps.
DOI: 10.5220/0005726906480655
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 648-655
ISBN: 978-989-758-168-7

2 RELATED WORK

When it comes to MDD of web applications and especially RIA, several approaches have been proposed in recent years that try to bring the paradigm of MDD into the world of web engineering and highly interactive user interfaces (Pleuss et al., 2011; Sauer, 2011). One is UWE (UML-based Web Engineering) (Kraus et al., 2007; Kraus, 2007). For generating the UI, UWE contains a presentation model in which the basic layout and functional structure of the UI can be modeled using abstract elements, for example buttons or text input fields. These elements must then be mapped to concrete UI components of the target platform. Also its application to specific web development platforms like UWE4JSF using Java ServerFaces (JSF) (Kroiss et al., 2009; Kroiß, 2008) or ASP.NET (Barabás and Kakucs, 2014) has been demonstrated. (Koch et al., 2009) uses UML state machines in order to describe patterns for the interaction of UI components triggered by user or system events, like e.g. onfocus or mouseover. These patterns can be integrated into models of all UML-based web methodologies. As a proof-of concept, they choosed UWE and extended the presentation layer meta-model to add the patterns to UI components using meta-attributes. However, these patterns only cover the interaction with the UI and not the components themselves. Moreover, the RIA patterns are not generated automatically but have to be translated into code manually.

A different approach extends the WebML standard[2] for modeling RIA (Bozzon et al., 2006a; Bozzon et al., 2006b). It focuses in particular on the distribution of functionality and data between the server and the client as well as on the data model and business logic. Regarding the UI, only the structure of the page can be modeled, but not single UI components.

With WebRatio a WebML based tool exists (Brambilla and Fraternali, 2014), which is also available as a commercial product. It was first introduced in 2001. Since then the tool, as well as the underlying modeling language, have been under further development. Nonetheless, WebRatio also focuses on data models and business logic and lacks capabilities of modeling UI components.

(Brambilla et al., 2014) describes an extension to the Interaction Flow Modeling Language[3] (IFML) for MDD of mobile applications based on HTML5, CSS and JS, optimized for the Apache Cordova Framework. IFML was developed by WebRatio and is inspired by their experience of WebML and the WebRatio tool. It has been adopted as a standard by the OMG and can be used to model the structure and content of web pages, as well as the users interaction for navigating from one page to another. However, as stated on the web site, "IFML does not cover the modeling of the presentation issues (e.g. layout, style and look&feel) of an application front-end and does not cater for the specification of bi-dimensional and tridimensional computer based graphics, video games, and other highly interactive applications."[4]. (Linaje et al., 2009) combines WebRatio for data models and business logic with the RUX-Method (Linaje et al., 2007b) for the presentation layer, which focuses more on the graphical UI than WebML. (Linaje et al., 2010) also uses this combination and additionally integrates context-awareness into both methods (e.g. time-aware presentation, location-aware services). As described in (Linaje et al., 2008), the RUX-Method breaks down the UI into three interface levels: Abstract, Concrete and Final Interfaces. Each level is build using components specified in a components library. But also RUX is lacking the capability of modeling the components themselves. It has also been combined with UWE for data and business logic (Preciado et al., 2008). Other approaches focus on migrating legacy web applications to RIA using WebML (Rodriguez-Echeverra et al., 2010) and RUX (Linaje et al., 2007a).

Many of the above approaches use templates or libraries in terms of UI generation and none of them cover the model-driven development of the UI components themselves. In contrast, (Huber and Brune, 2012) focuses on the MDD of interactive UI components using the canvas element of the recently announced HTML5 standard. However, the used metamodel is rather simplistic and uses relatively primitive graphical elements. For developing more complex applications using MDD, a higher level abstractions is required.

Since JS has taken such an important role in web engineering, many frameworks and libraries have been developed to simplify and speed up the development of websites with JS. Most noteworthy jQuery[5], which offers a wide variety of features like DOM manipulation, event handling, animation, and in particular a very fast and easy way to embed AJAX (Asynchronous JavaScript and XML) functionality. In addition to the standard jQuery library there exist jQery UI and jQuery Mobile. While many jQuery widgets are developed by third party users and therefore not guaranteed to be maintained and developed any fur-

[2] www.webml.org
[3] www.ifml.org
[4] www.ifml.org
[5] http://www.jquery.com/

ther, the Dojo Toolkit[6] offers similar functionality and comes with a built-in subproject for UI widgets.

jQuery and Dojo have been evaluated along with other JS frameworks regarding their quality of code (complexity and maintainability), vulnerability, and performance (Gizas et al., 2012) as well as with respect to multimedia support (Rosales-Morales et al., 2011).

Another framework offering great responsiveness and therefore qualifying for the development of mobile applications, is the Bootstrap[7] framework. It was developed by Twitter and is currently the most popular project on GitHub[8]. It's built-in layout functionality uses a twelve column grid, that adapts to the users viewport. If the screen is too narrow, the columns become fluid and are stacked vertically, in order to avoid horizontal scrolling even on small screens.

Still in an early stage of development is the Famo.us framework (Famo.us, 2014) with the distinctive feature of a built in 3D layout engine and physics animation engine, capable of rendering to DOM, Canvas, or WebGL. It can also be used for rendering in applications, developed with the AngularJS[9] framework and will be integrated into the next generation of jQuery widgets (Borins, 2015). Despite its superior 3D graphics capabilities, it has not been evaluated so far with respect to its usage for developing interactive graphical UI components.

As one can see, many approaches exist in the scientific literature as well as in practice for MDD of client-server web applications or for simplifying JavaScript development by using frameworks. However, few results have been proposed for MDD of JavaScript UI components using these frameworks. In particular, MDD of highly interactive JavaScript components for implementing virtual world scenarios has not been studied so far. Therefore, in the following the question is addressed how JavaScript UI components representing virtual world scenarios could be realized using MDD and the Famo.us framework.

3 DESIGN OF THE UML MODEL

The proposed UML class model describes a UI component representing a semi-3D scenery consisting of a boardgame-like "stage" (2D grid), which represents the virtual world and on which objects are either fixed to a position or could move across the fields. For higher interaction, actions can be added to the objects user for changing the object's properties, e.g. animation speed, size, or opacity. Specific fields can also be marked as an exit for redirecting the user to another setting (i.e. a new scenery or game level).

Ecore serves as the modeling language. It is used to define a model for the target domain, which is designed using the Ecore Diagram Editor. The instance models for the code generation are then created with the EMF integrated editor. This editor offers a tree based structure for modeling and automatically generates an XMI file representing the modeled instance. Ecore was chosen as modeling language since it is free, open-source, and can be used with the popular Eclipse IDE. It also offers all the needed modeling elements like classes, attributes, references and inheritance.

The proposed UML class model is shown in figure 1. Its structure is guided by the idea of representing a specific scenery (representing i.e. a room or level of a virtual world) in a stage-like fashion by a rectangular field made up of quadratic cells (like e.g. on a chess board), each adressable by position (x,y)-coordinates. The field represents the ''floor" of the scenery. On this field, objects (representing characters, scenery items ec.) could be placed and moved around.

The root element of the metamodel is the *Application* class in the top left. An *Application* instance represents a set of *Field* objects (representing levels or sceneries). It does not have any attributes and only serves as a container for the other elements. Without the *Application* class a new model instance would have to be created for each field and the generator would have to be executed for each resulting model file. Therefore the *Application* class makes the development process more efficient when creating more than one field at once. An *Application* consists of one or more fields.

The *Field* class is the main class of the model and represents the ''floor" of the scenery, i.e. the light blue, two-dimensional plain in figure 2. It has a *name* attribute, which will later be the name of the folder in which the generator puts the source code for the field. The *surfaceSize* specifies the size of each square on the field. The *color* attribute defines the color and can be in hexadecimal (e.g. #AA00FF) or rgb (e.g. rgb(176,0,255)) format. The next three attributes, *borderWidth*, *borderStyle*, and *borderColor*, determine the border of each square and have to be formatted like CSS border values. *tiltX*, *tiltY*, and *tiltZ* are used for tilting the field, to give it a three dimensional effect. The last attribute of the *field* class is called *content* and indicates the image that should be used as a background for each square.

[6]http://dojotoolkit.org
[7]http://www.getbootstrap.com
[8]https://github.com/
[9]https://angularjs.org/

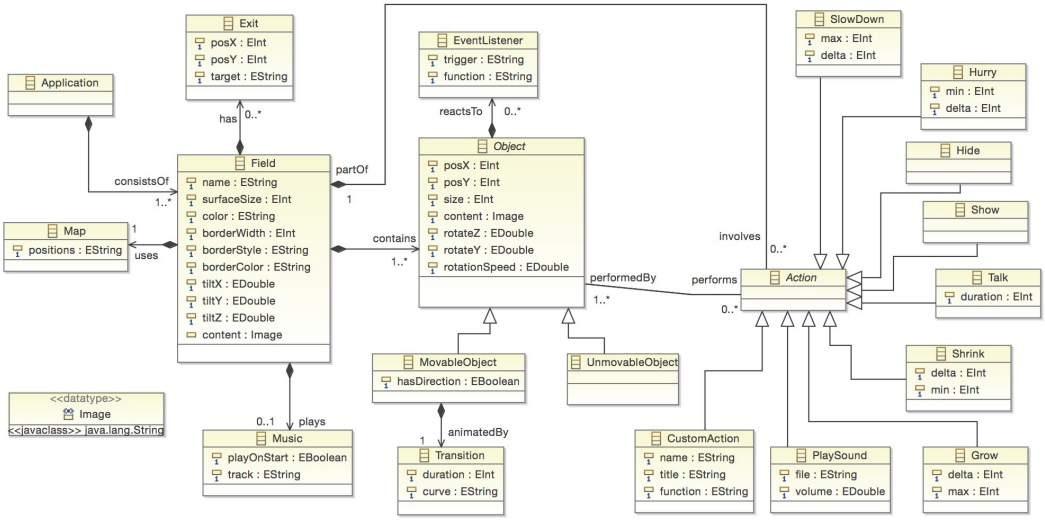

Figure 1: Proposed UML class model describing the UI component for a virtual world scenario.

Each field uses a *Map* instance, which specifies the size and shape of the field. The upper left corner is position [0,0], from there on x increments to the right and y downwards. The format of the attribute *positions* has to be a string of 1s and 0s, separated by commas. A 1 indicates that this position is part of the field and a square will be drawn at this position. A 0, on the other hand, indicates that this position should be left out. The String will later be transformed to an array. A field is always quadratic and its size is determined by taking the square root of that array. For example the string *"1,1,1, 1,0,1, 1,1,1"* results in a 3×3 field with an empty space in the middle. If the number of provided positions is not quadratic, a smaller quadratic field is created and the additional positions are ignored. For example if the positions string contains 17 values, a 4×4 field is created and the last positions is ignored.

With the *Music* class, a background music can be added to a field. The attribute *track* represents the name of the sound file and the boolean value *playOnStart* indicates if the music should play right away upon display of the field. The file can be of any format supported by the HTML5 <audio> element.

The *Exit* class is used to redirect the user to another field, when he or she moves an object to the position of the exit. *PosX* and *posY* specify the position of the exit on the field. The string *target* defines the name of the field to which the user should be redirected.

The *Object* class represents objects on the field (characters and other items). A field contains at least one object and a maximum of one object per available position. *Object* is an abstract class and serves as a parent for *MovableObject* and *UnmovableObject*, containing all their common attributes. *posX* and *posY* indicate the initial position of the object on the field. If the position doesn't exist, or is already occupied by another object, the object is ignored. The *size* attribute determines its size and *content* the image that serves as a visual representation on the field. *rotateZ* and *rotateY* are used to rotate the object. These values are in radians, so the input of π for example, will result in a rotation of 180 degrees. An object can also be animated by specifying a *rotationSpeed* other than 0. This will result in an object continuously spinning around its y-axis. A higher value results in a faster animation.

The class *UnmovableObject* doesn't require any additional attributes. *MovableObject* on the other hand contains a boolean value called *hasDirection* to define whether the content of its image is looking in a certain direction. If this is set to true, the image will be turned around in the direction the object is moving.

A *MovableObject* is animated by a *Transition*. The *duration* attribute determines the duration of the animation and the *curve* attribute indicates a curve from the Famo.us *Easing* class.

A listener pattern is used to implement the reactions to the events triggered by movin onto certain fields or interacting with other objects. The *EventListener* class was designed for adding reactions to certain events. Therefore, a reaction has to be implemented in the *function* attribute and a *trigger* has to specify when this function should be executed.

The *Action* class offers ways the user can interact with the objects, besides moving them across the field. It is an abstract class and has two relations. The first relation implies that an action is performed by an object. The second relates an action to a field. This

second relation enables an action to be performed by multiple objects. As a consequence, e.g. if three objects perform the same action, it only has to be modeled once.

There are several predefined actions. *SlowDown* increases the duration of an object's transition and therefore slowing down its movement. *delta* defines the amount by which the duration is increased and *max* specifies the maximum for the duration value.

Hurry implements the opposite effect. Other similar actions are the *Grow* and *Shrink* actions, but instead of altering the object's movement they increase or decrease its size.

The *Talk* action is used for taking a text input from the user and displaying it in a rectangle next to the object for as many milliseconds as supplied in the *duration* attribute.

For audiovisual interaction the *PlaySound* action can be added to an object. When executed the file defined in the *file* attribute will be played. Moreover, the *volume* can be set to a value between 0 and 1.

Finally, the *CustomAction* class allows the developer to add manually written functions. Besides the *function* itself, a *name* and a *title* for the action have to be defined. The *name* attribute uniquely identifies the action and the *title* specifies the action's title. An action can be combined with the *EventListener* to enable objects to interact among themselves.

4 MODEL TRANSFORMATION AND CODE GENERATION

The generator, which transfers these models to executable code, is implemented in Java, which was chosen since the generator should be easily integratable in the Eclipse IDE framework (written also in Java) and the Java ecosystem provides well established libraries to support the necessary processing steps.

The generator makes use of the *DocumentBuilder* class from the package *javax.xml.parsers* to create a DOM object from the XMI file. The DOM object is then parsed using the *NodeList* and *Element* classes from the *org.w3c.dom* package. The result of the generation process is a JavaScript application which can be viewed in a browser by opening a generated index.html file.

Sound files and images used by the modeled application have to be copied either into the sounds respectively graphics folder within the source folder in advance or in respective folders of the target directory after the generator has been executed.

The generated JavaScript code uses a custom JavaScript class library which encapsulates the Famo.us functionality and provides classes corresponding to the classes defined in the UML model as far as necessary.

5 PROOF-OF-CONCEPT IMPLEMENTATION

To demonstrate the feasibility of the proposed approach, a prototype virtual world scenario was modeled and generated. The resulting JavaScript application generated by the code generator is shown in the screenshot in figure 2. To generate this scenario, first an instance of the UML model of figure 1 was created using the built-in Eclipse EMF editor. This editor automatically creates a XMI file and visualizes the content in a tree-like structure. Values for the attributes of the currently selected element are entered using the properties view in Eclipse. Additionally, the editor offers live validation of the models and indicates missing attributes or elements immediately.

Figure 3 shows a screenshot of the tree-like structure of the prototype instance model within the Eclipse EMF editor. The first element to be added for each application is a field, representing the scene. For each field, a map is added to define the shape and size of the field. For the prototype, the result will be a field with 6×6 positions, but only the three inner positions of each row will be accessible. An exit to another field is added at position [3,3]. Moreover, a background music will start playing once the browser has loaded the page containing the field.

The protoype field contains four objects. One unmovable and three movable objects. The last object also includes an event listener which is executed whenever the user uses the talk function of another object with the input *"Hello Partner!"*. When modeling event listeners, their parent object can be referenced by the keyword *thisObject*. The generator will automatically replace the keyword with a reference to the parent object. If elements of a subclass are added, the type is added as an attribute to distinguish between the different subclasses. The actions of the element are specified by two indexes. The first one indicates in which field node of the application the action was defined and the second one is the index of the action itself inside of that field node. This means that it is possible to use an action defined in one field for objects throughout the whole application.

The last step is to add the actions. For demonstration purposes a minimalistic custom action is also part of the prototype model. Since the relationship between action and object is bi-directional in the model, actions also have an attribute containing the objects

Figure 2: Screenshot of the prototype virtual world scenery generated from the modeled instance.

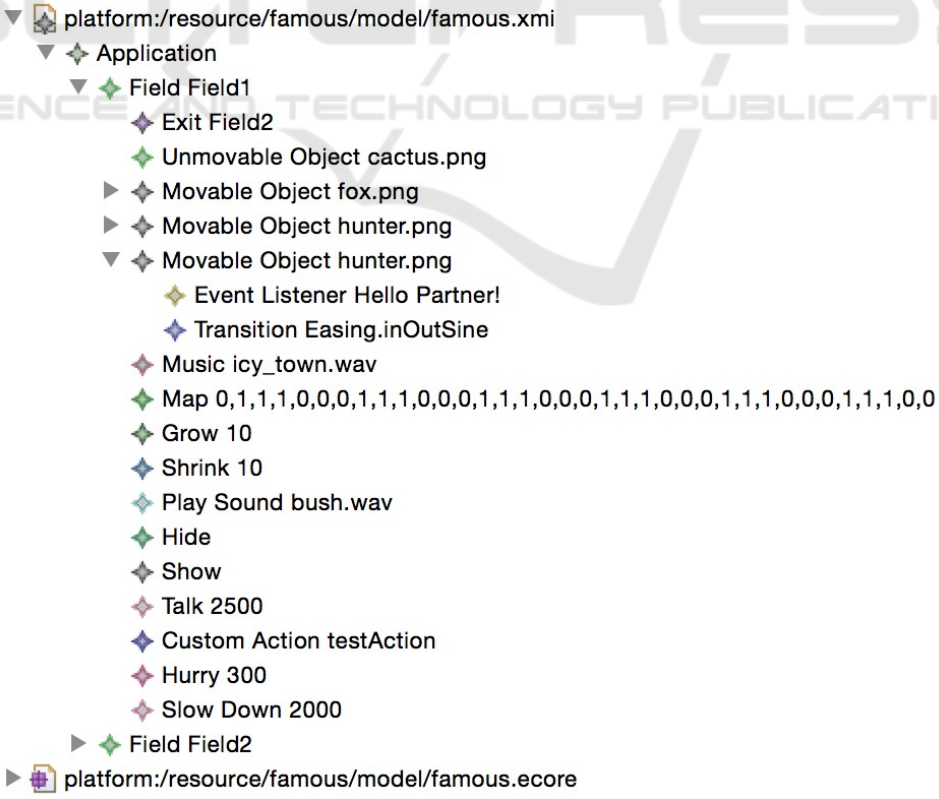

Figure 3: Tree-like representation of the model instance in the Eclipse EMF editor.

that perform them.

To further evaluate the approach, the prototype model was extended by a second field. To demonstrate the interaction with the user, several movable objects were added to the field. Moreover multiple actions are performed by the individual objects, letting the user modify their appearance. For objects containing the *talk* action, the user can enter text into the input field in the top left, which is then displayed in a dialogue box next to the selected object. An event listener has been added to one of the objects, replying automatically if the user enters a certain text. The field also includes a background music, which can be played by clicking on the speaker icon in the top right corner. Sound can also be played by objects, when selecting the *Play Sound* entry in their action view.

The time needed for creating this field was significantly lower compared to a manual implementation. It was also faster than copying the code of one field and then altering it to design a new one. The main reason for the reduced time effort is the much higher abstractions that are used for development. Additionally, the use of the tree-based Eclipse editor also increases the development speed, especially when reusing actions for several objects. The live model validation offers early error detection, so no time is wasted with failed generator runs. However, there are also some drawbacks in the development process. The properties view of Eclipse, which is used to enter the values for the attributes is not suited for comprehensive inputs like i.e. long arrays or functions. Moreover, the Ecore feature of automatically setting default values and leaving out attributes in the XMI file made the implementation of the generator more challenging.

6 CONCLUSIONS

In conclusion, in this paper an approach for implementing JavaScript UI components representing virtual world scenarios using MDD was proposed. Using the Ecore modeling language, a metamodel containing classes for modeling a semi-3D game-like environment and the corresponding code generator were developed. While using high abstractions for the modeled classes, the *EventListener* and *CustomAction* classes also allow the developer to manually add functionality.

The code generator was implemented in Java for transforming model instances into executable JS code. The underlying JS codebase was developed with the upcoming Famo.us framework, offering sophisticated animations for the UI components.

The approach was evaluated by a proof-of-concept implementation demonstrating its general feasibility, revealing the advantages and disadvantages of the Eclipse-based model editor. However, further research is needed to evalaute the approach in more complex scenarios and also in real-world case studies and to analyze its integration with the business logic tier and back-end systems.

REFERENCES

Barabás, L. and Kakucs, B. (2014). Model-driven development of web applications. *International Journal on Recent Trends in Engineering & Technology*, 10(1).

Borins, M. (2015). Famo.us partners with the jquery foundation.

Bozzon, A., Comai, S., Fraternali, P., and Carughi, G. T. (2006a). Capturing ria concepts in a web modeling language. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 907–908, New York, NY, USA. ACM.

Bozzon, A., Comai, S., Fraternali, P., and Carughi, G. T. (2006b). Conceptual modeling and code generation for rich internet applications. In *Proceedings of the 6th International Conference on Web Engineering*, ICWE '06, pages 353–360, New York, NY, USA. ACM.

Brambilla, M. and Fraternali, P. (2014). Large-scale model-driven engineering of web user interaction: The webml and webratio experience. *Science of Computer Programming*, 89:71–87.

Brambilla, M., Mauri, A., and Umuhoza, E. (2014). Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. In *Mobile Web Information Systems*.

Famo.us (2014). Famo.us layout.

Gizas, A., Christodoulou, S., and Papatheodorou, T. (2012). Comparative evaluation of javascript frameworks. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 513–514, New York, NY, USA. ACM.

Huber, M. and Brune, P. (2012). Model-driven development of interactive web user interfaces with html5. *University of Applied Sciences Neu-Ulm*.

Koch, N., Pigerl, M., Zhang, G., and Morozova, T. (2009). Patterns for the model-based development of rias. In Gaedke, M., Grossniklaus, M., and Díaz, O., editors, *Web Engineering*, volume 5648 of *Lecture Notes in Computer Science*, pages 283–291. Springer Berlin Heidelberg.

Kraus, A. (2007). *Model driven software engineering for web applications*. PhD thesis, Ludwig-Maximilians-Universität München.

Kraus, A., Knapp, A., and Koch, N. (2007). Model-driven generation of web applications in uwe. *MDWE*, 261.

Kroiß, C. (2008). Modellbasierte generierung von web-anwendungen mit uwe. *Ludwig-Maximilians-Universität München*.

Kroiss, C., Koch, N., and Knapp, A. (2009). Uwe4jsf: A model-driven generation approach for web applications. In Gaedke, M., Grossniklaus, M., and Diaz, O., editors, *Web Engineering*, volume 5648 of *Lecture Notes in Computer Science*, pages 493–496. Springer Berlin Heidelberg.

Linaje, M., Preciado, J., Morales-Chaparro, R., Rodriguez-Echeverra, R., and Sánchez-Figueroa, F. (2009). Automatic generation of rias using rux-tool and webratio. In Gaedke, M., Grossniklaus, M., and Díaz, O., editors, *Web Engineering*, volume 5648 of *Lecture Notes in Computer Science*, pages 501–504. Springer Berlin Heidelberg.

Linaje, M., Preciado, J., and Sanchez-Figueroa, F. (2007a). Engineering rich internet application user interfaces over legacy web models. *Internet Computing, IEEE*, 11(6):53–59.

Linaje, M., Preciado, J., and Sánchez-Figueroa, F. (2007b). A method for model based design of rich internet application interactive user interfaces. In Baresi, L., Fraternali, P., and Houben, G.-J., editors, *Web Engineering*, volume 4607 of *Lecture Notes in Computer Science*, pages 226–241. Springer Berlin Heidelberg.

Linaje, M., Preciado, J. C., Morales-Chaparro, R., and Sanchez-Figueroa, F. (2008). On the implementation of multiplatform ria user interface components. In *7Th International Workshop On Web-Oriented Software Technologies–IWWOST*, volume 8, pages 50–55.

Linaje, M., Preciado, J. C., and Sánchez-Figueroa, F. (2010). Multi-device context-aware rias using a model-driven approach. *Journal of Universal Computer Science*, 16(15):2038–2059.

Pleuss, A., Gračanin, D., and Zhang, X. (2011). Model-driven development of interactive and integrated 2d and 3d user interfaces using mml. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 89–92, New York, NY, USA. ACM.

Preciado, J., Linaje, M., Morales-Chaparro, R., Sanchez-Figueroa, F., Zhang, G., Kroiss, C., and Koch, N. (2008). Designing rich internet applications combining uwe and rux-method. In *Web Engineering, 2008. ICWE '08. Eighth International Conference on*, pages 148–154.

Rodriguez-Echeverra, R., Conejero, J., Linaje, M., Preciado, J., and Sanchez-Figueroa, F. (2010). Re-engineering legacy web applications into rich internet applications. In Benatallah, B., Casati, F., Kappel, G., and Rossi, G., editors, *Web Engineering*, volume 6189 of *Lecture Notes in Computer Science*, pages 189–203. Springer Berlin Heidelberg.

Rosales-Morales, V., Alor-Hernandez, G., and Juarez-Martinez, U. (2011). An overview of multimedia support into javascript-based frameworks for developing rias. In *Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference on*, pages 66–70.

Sauer, S. (2011). Applying meta-modeling for the definition of model-driven development methods of advanced user interfaces. In Hussmann, H., Meixner, G., and Zuehlke, D., editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 67–86. Springer Berlin Heidelberg.

Stangl, B., Kastner, M., and Polsterer, F. (2012). Social virtual worlds' success factors: Four studies' insights for the tourism supply and demand side. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 993–1002.