The Exploitation of OpenAPI Documentation for the Generation of Web Frontends

István Koren, Ralf Klamma
RWTH Aachen University
Aachen, Germany
koren,klamma@dbis.rwth-aachen.de

ABSTRACT

New Internet-enabled devices and Web services are introduced on a daily basis. Documentation formats are available that describe their functionalities in terms of API endpoints and parameters. In particular, the OpenAPI specification has gained considerable influence over the last years. Web-based solutions exist that generate interactive OpenAPI documentation with HTML5 & JavaScript. They allow developers to quickly get an understanding what the services and devices do and how they work. However, the generated user interfaces are far from real-world practices of designers and end users. We present an approach to overcome this gap, by using a model-driven methodology resulting in state-of-the-art responsive Web user interfaces. To this end, we use the Interaction Flow Modeling Language (IFML) as intermediary model specification to bring together APIs and frontends. Our implementation is based on open standards like Web Components and SVG. A screencast of our tool is available at https://youtu.be/KFOPmPShak4

CCS CONCEPTS

• **Information systems → RESTful web services**; • **Software and its engineering → Integrated and visual development environments**; *Design languages*;

KEYWORDS

OpenAPI; IFML; Web Components; Interaction Design

ACM Reference Format:
István Koren, Ralf Klamma. 2018. The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3184558.3188740

1 INTRODUCTION

Our society is currently experiencing an unprecedented introduction of new Internet-enabled device types that come in various sizes and shapes. Examples reach from laptops and smartphones to wearable augmented reality glasses. Nearly every consumer device is able to communicate with Web resources over TCP/IP and HTTP(S). Additionally, consumer devices like smartphones, tablets and even smart watches feature Web browsers displaying interactive HTML5 pages with JavaScript. The challenges in developing applications supporting these devices are manifold. On the backend scalability, reliability and maintainability issues are addressed by the componentization of software into virtualized containers running microservices. These services expose application programming interfaces (API) that can be used by developers to create applications. To simplify the usage of APIs, a number of API documentation standards have been proposed; for REST-based Web services, the OpenAPI specification [10], formerly known as Swagger, became the apparent market leader. An ecosystem of related toolsets has evolved around the specification offering code generation and simple Web frontends. While these tools are developer-friendly, they are not really intended to be used by non-developers like designers and end users. For example, generated user interfaces contain many technical details like JSON schemas and possible error codes. To this end, our main goal is to automate the time-consuming process of user interface creation from API to Web frontend. We want to be able to generate application prototypes whose design can be tailored to user-specific demands in a consecutive step. By that, we additionally free up developer resources to address further user requirements.

In this article, we present a tool for generating and designing Web frontends based on the OpenAPI documentation format. It is able to communicate with various Web services through their APIs. We aim for a collaborative platform where the whole community consisting of domain experts, designers and developers can work together on an application. To make interactions as easy as possible, drag & drop behavior and tooltips are essential functional requirements. On the non-functional side, the main goal is to entirely rely on established Web standards, or at least open specifications supported by various parties. To account for the large number of display form factors, we follow a model-driven interaction design approach with the help of the Interaction Flow Modeling Language (IFML). Conceptual models are familiar to developers from software engineering and database design; IFML in particular can be used to design device-independent interactions between frontend components. Our implementation is built upon the recent Web Components group of W3C specifications, to allow modularity and reusability across applications.

The remainder is structured as follows. Section 2 presents related research in the area of generating Web frontends based on service descriptions. Section 3 explains related Web technologies. The conceptual design is explained in Section 4. In Section 5 we explain the implementation details. Section 6 discusses strengths and weaknesses of our approach. Section 7 concludes our article with an outlook on future work.

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3188740

2 RELATED WORK

Regarding cross-device user interfaces, research reveals that several approaches have been made to create universal user interface descriptions applicable to various application areas. The most notable endeavor targeting the Web is the Cameleon reference framework [5]. It defines a multi-step procedural model from task models over abstract UI specifications to a concrete UI.

Maximilien et al. introduce *Swashup*, a domain-specific language for Web API and service mashups [16]. It leverages service descriptions like WSDL but focuses on the service composition. Similarly, He and Yen [8] and the *ServFace Builder* [17] rely on WSDL files. The ServFace approach generates user interfaces, but is driven by annotations written by the service developer. Vaziri et al. generate chatbots out of Swagger documentation to interact with APIs in natural language [21]. Their system is able to improve the specification by letting users interact with the chatbot. The authors report that many REST APIs feature complex JSON structures, while their chatbot is designed for scalar input such as strings and dates. Swagger is also used by [13] that suggests the user APIs based on contextual properties. For each service, an Android user interface is generated; the goal is to provide uniform interfaces for a wide variety of context-dependent services.

There are several approaches that connect Internet of Things (IoT) device APIs to the Web. *Simurgh* uses the RAML API specification language to discover and integrate Internet of Things (IoT) devices [12]. Their functionalities are then combined by end users. Software products in the area of multi-vendor aggregation of IoT devices are for example Node-RED [6] and IFTTT [9]. However, while offering powerful features and a great user experience, these tools are neither multi-user capable for synchronous collaboration nor based on a vendor-independent, standardized notation.

3 WEB TECHNOLOGIES

In the following, we introduce the standards and Web technologies behind our tool. First, the OpenAPI specification is presented, then the Interaction Flow Modeling Language (IFML) gets highlighted. Finally, we show the W3C Web Components group of standards.

3.1 OpenAPI Specification

OpenAPI is an application program interface (API) documentation specification for RESTful Web services [10]. It allows creating machine-readable interface descriptions for documenting, producing, consuming and visualizing APIs based on HTTP. Formerly known as Swagger, it enjoys widespread support in the Web community, with various open source projects built around it. The standardization is lead by the Open API Initiative under the hood of the Linux foundation; it was founded in late 2015. Major companies like Google, IBM and Microsoft contribute to the development of the specification. The *APIs.guru* website alone currently lists around 550 publicly available APIs with a Swagger/OpenAPI documentation[1]. OpenAPI documents describe an API in either the YAML or JSON markup language; both formats can be translated into each other without loss of information.

In the listing below, an example OpenAPI document is provided in YAML format. It documents a simple address book Web service with `/contacts` as single resource and methods for retrieving and deleting contacts. Particular contacts can be retrieved by supplying an ID as path parameter. Table 1 gives a non-comprehensive overview of the main concepts behind the properties of an OpenAPI document.

```yaml
---
openapi: 3.0.0
servers:
- description: Development Server
  url: http://127.0.0.1:3000
info:
  version: 1.0.0
  title: Address Book Service
  description: The API of the Address Book Service.
tags:
- name: contact
  description: Everything about contacts.
paths:
  "/contacts":
    get:
      tags:
      - contact
      description: Returns all contacts.
      operationId: getContacts
      responses:
        '200':
          description: All the contacts.
          content:
            application/json:
              schema:
                type: array
                items:
                  "$ref": "#/components/schemas/Contact"
  "/contacts/{contactId}":
    get:
      tags:
      - contact
      description: Returns a particular contact.
      operationId: getContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
      responses:
        '200':
          description: A specific category.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/Contact"
    delete:
      tags:
```

[1] https://apis.guru/openapi-directory/

Table 1: OpenAPI Properties

Property	Description
openapi	OpenAPI version
info	title and version of the document
servers	list of concrete endpoint URLs
security	authentication and authorization options
paths	list of API paths, prefixed by server URL
tags	groups paths into categories
components	reusable details like parameter schemas

```
      - contact
      description: Deletes a contact.
      operationId: deleteContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
      responses:
        '200':
          description: Contact deleted.
        '404':
          description: Contact not found.
components:
  schemas:
    Contact:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        lastname:
          type: string
        email:
          type: string
```

Swagger UI [20] is an open source software that automatically generates a HTML5-based visualization and interaction frontend from an OpenAPI file. The paths are presented as lists which are grouped by tags. Interaction capabilities are very limited, in the sense that developers can enter parameters into text boxes to test client requests and server responses. Figure 1 shows a clipped screenshot of the HTML documentation of the above mentioned address book example, as generated by Swagger UI.

3.2 Interaction Flow Modeling Language

The Interaction Flow Modeling Language (IFML) is a visual domain-specific modeling language for the visual design of abstract user interactions and data flows within user interfaces [4]. Developed in 2012 and 2013, it is standardized by the Object Management Group

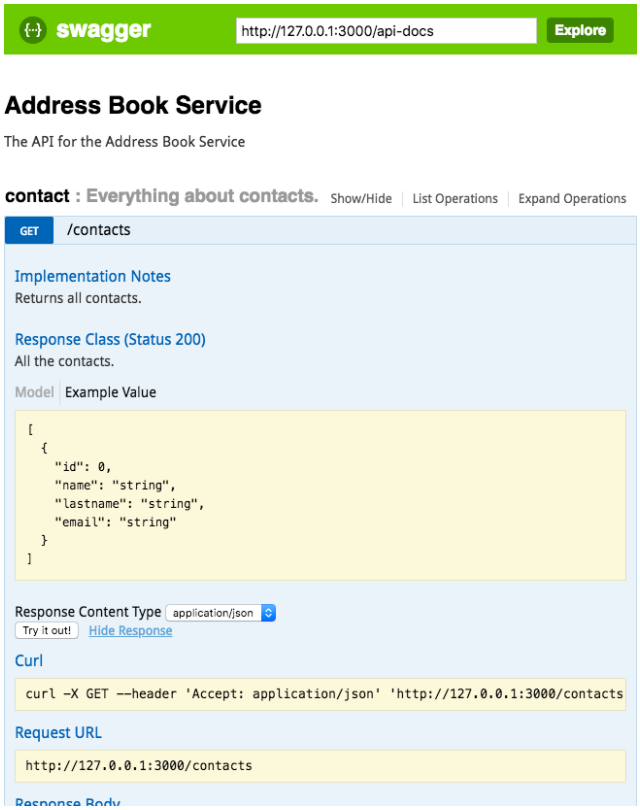

Figure 1: Address Book Example in Swagger UI

(OMG), the same standardization body that is also governing the UML standard prevalent in software engineering. An IFML model is a platform-independent representation of a graphical user interface on devices such as mobiles, laptops or wearables. Figure 2 showcases the IFML model of the very simple address book application. It displays a list of contacts; upon selecting a particular contact, the details are shown in a view right next to the list. Below the details, a button allows deleting the selected contact. The concepts behind the model elements are explained in the following.

An IFML model has one or more possibly nested *View Container* at its root ("Address Book"). A View Container can represent for instance an application window of a desktop application, or the main view of a Web page. It can contain *View Components* standing for certain user interface elements like a list or a table ("Contacts List" and "Contact Details"). These model elements can be associated with *Events* ("selected" and "delete") or *Actions* (upon selecting the "delete" button). An Event can be generated by the user or system, for instance when clicking a button. It can trigger an Action, for example the transition of one view to another. State transitions and data flows are modeled as edges with *Navigation Flows* or *Data Flows*. Navigation Flows are happening upon user interaction; it then represents the transition from event to another view. Data Flows represent any data passing from one element to another, for example, the identifier of a selected menu item.

While the standard is platform-independent, it can be used to generate concrete user interfaces. The standard document contains

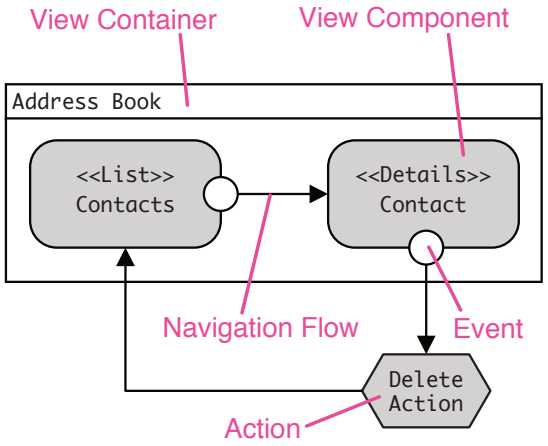

Figure 2: Address Book IFML Model

multiple example mappings to user interface description languages, for instance to the Windows Presentation Foundation, Java Swing, and HTML. For example, a View Container may be translated to an HTML `<div>` element, and the nested View Components to respective child elements.

The official IFML website lists several commercial and open source model editor implementations. Most notable, we refer to IFMLEdit.org, a Web-based modeling tool able to generate Web clients out of IFML models [3].

3.3 Web Components

A Web Component is a custom, reusable DOM element that can be used in the DOM like a native HTML element. The Web Component group of specifications is currently being standardized to be part of the HTML and DOM specifications. The *Custom Element* feature adds new methods to the DOM to register new elements together with their identifier that needs to contain a dash in order to differentiate existing, native tags like `<div>` and `<p>`. The *Shadow DOM* encapsulates the scope of CSS styles and JavaScript code to the internal children of a node. While these features are being implemented in all major Web browsers, JavaScript *polyfills* ensure backward compatibility of these APIs.

Web Components are a recent manifestation of the increased adoption of component-based software engineering in the frontend Web. Similar approaches on a JavaScript framework level are Ember, React and Vue.js. In our implementation, we use the Google-maintained Polymer library in version 2.0. It uses features of the ECMAScript 6 language version like classes and mixins. Advanced custom elements based on Polymer simplify *Service Worker* registration and caching, making Web applications possible that largely work without an active Internet connection. Polymer also comes with high-fidelity pre-designed template elements following the *Material Design* guidelines [7]. They achieve high accessibility and a familiar user experience across device types and display form factors.

4 TRANSFORMATION APPROACH

After presenting the employed standards and technologies, we describe the unter interface generation approach in this section. An overview of the generation sequence from OpenAPI specification over an IFML model to a concrete user interface based on HTML5 & JavaScript is given in Figure 3. The figure highlights concepts that are transformed into each other with the same background color (from left to right). In the OpenAPI specification, the URL of a server endpoint is encoded through the combination of the server property and an item in the paths array (purple/dark highlight). This information is encapsulated in the Action of the IFML model. The HTML5 & JavaScript instance contains the URL information in the method that is called when submitting the form. The form itself is represented by a View Component in the IFML model (yellow/light background). It originates in the parameters property of the endpoint. In the following, we present the above process in detail and present further mappings between the API specification, modeling and HTML5 & JavaScript implementation concepts.

4.1 API Documentation to Model

The first step in the user interface generation sequence involves creating an IFML model from an OpenAPI document. The resulting IFML model helps to abstract away platform-specific implementation details. Two types of transformation are already presented in Figure 3. The server URL and the path properties of the OpenAPI document is an integral part of every transformation that involves data exchange between client and server. The combination of both defines the server endpoint, for instance, to retrieve content or to upload input from the client. While this information is relevant for the later transformation to user interface code, it is not visualized in the model. We therefore add semantic annotations to the model. Another transformation type is generating View Components from the parameters array of the documentation. OpenAPI follows the JSON schema description language. It describes a JSON data structure and allows its automated validation. The two main data types used to build structural hierarchies are objects and arrays. Objects are a key-value collection of entries, while arrays are a list of items. The key of an object is a string type, while the value and array items can be any of *object*, *array*, *string*, *number*, *boolean*, or *null*. Here, the first level of the data type hierarchy is of interest. If it is an array, the generated model element is a *List*; objects are generated as *Form* and *Detail* View Components. Other types, e.g. a REST endpoint returning a single string, are transformed into plain View Components.

4.2 Model to Frontend

In the second step, the generated IFML models are transformed into the concrete user interface based on HTML5 & JavaScript. The basic principle behind representing IFML concepts with HTML elements is described in the official IFML specification annex E [19]. There, the whole model is mapped to a *WebSite* element, which represents the main `<body>` tag of an HTML page. Other mappings are for instance View Containers as `<div>` tags, Form View Components as equivalent `<form>` elements, and List View Components as `<table>` tags.

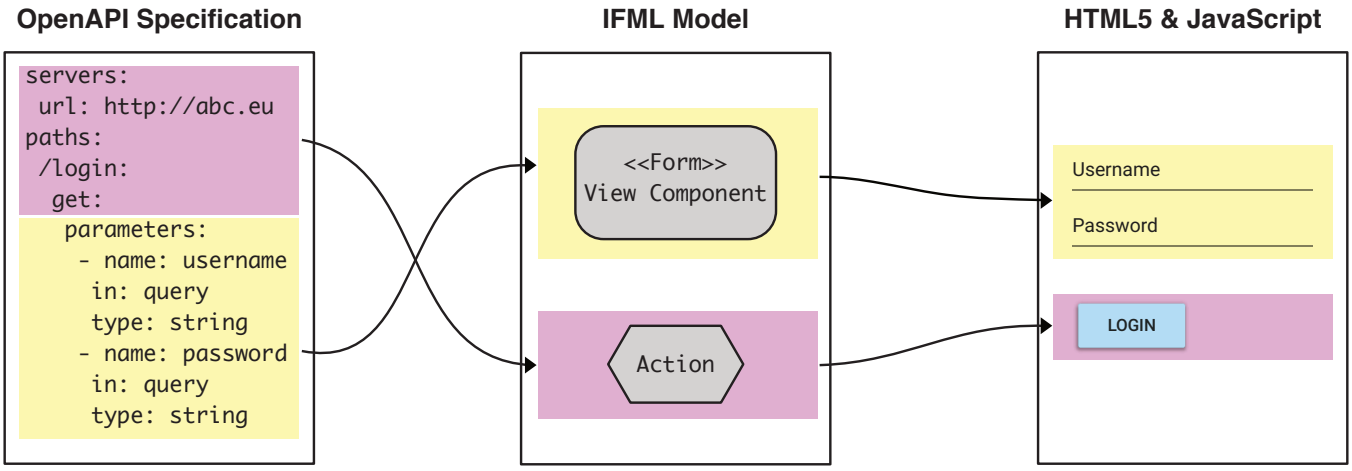

Figure 3: Left to Right: Transformation of OpenAPI Specification (simplified) over IFML to HTML5 & JavaScript

We leverage these mappings, but also use the metadata added to the model elements to generate JavaScript code, e.g. for submitting form inputs to the correct server endpoint. To influence the final layout of the generated user interface, further annotations can be used, like the flow direction of children (vertical or horizontal). These are transformed into CSS style instructions.

5 IMPLEMENTATION

In this section, we describe the implementation details of our Web application. The app is built using HTML5 Web Components and follows the general usability and accessibility guidelines of Material Design, in particular for the generated user interfaces. This results in a familiar look & feel across different frontends.

Figure 4 presents a screenshot of our tool running in the Chrome browser. It shows the IFML model of the address book example known from Figure 2. For demonstration purposes, we developed a server backend and filled it with demo data. We additionally added a toolbar as View Component. On the main level, the application is split between a vertical menu bar on the left, and the main tool window on the right. In the screenshot, the *Interaction Flow Designer* is shown. Its layout is divided horizontally into three separate zones; sidebars on the left and right and the canvas in between. In the following, we explain each zone in detail.

5.1 Model Element Palette

On the left sidebar, the user can choose between a *store* of components and a tree-based *hierarchy* of the nested model elements. The store metaphor intends to build on the concept of an app store, where model elements can be dragged into the canvas. At the bottom of the left sidebar, an *Add Elements* button allows to reach the *OpenAPI Import Dialog*. To integrate arbitrary API-enabled Web services into our tool, it requires a URL to an API documentation. Therefore, a URL to an OpenAPI specification file can be entered in the dialog. Confirming the import triggers the first step of the user interface generation process (cf. Figure 3). The tool then reads in the specification file and generates both IFML modules and corresponding HTML elements based on our predefined mapping, and shares them in the application's synchronized data store. The generated model artifacts are entered into the Store's list, called the *palette*.

5.2 Interaction Flow Model Editor

From the palette, model elements are dragged into the main canvas in the middle of the Interaction Flow Designer. It shows the complete model and allows manipulation of elements, like changing the size or changing group inheritance. The canvas can be freely zoomed in and out via buttons on the canvas toolbar on the top. Both the canvas and the IFML model elements can be dragged around. To create an event, it can either be dragged in from the palette, or put in place after clicking the border of an element; when hovering the mouse cursor over the button, the event is already shown. The canvas is implemented with the help of a Scalable Vector Graphics (SVG) element in HTML. All model nodes and edges have a direct vector representation. The main advantage is the direct export capability as SVG file. Additionally, the SVG element supports the export as a PNG file. Therefore, generated models can be refined in external vector-based graphics applications, or be reused in third party documents or Web applications as pixel graphic. After selecting a model node with the mouse or by tap, its properties are shown in the *Properties* panel on the right.

5.3 Properties Browser

The screenshot shows the position, size and title attributes of the selected model element. Below the model element properties, the properties of the transformation result are shown. For instance, in the case of an IFML View Component, the CSS style attribute can be influenced here. Below the properties panel, the *HTML Preview* gives an impression of how the generated HTML interface looks like. It also contains a button to open the generated frontend in a new browser tab.

5.4 Collaboration Architecture

Synchronous collaboration in near real-time is achieved via the Yjs library [11]. Near real-time refers to the humanly recognizable threshold of around 100 ms. Yjs synchronizes arbitrary data types in

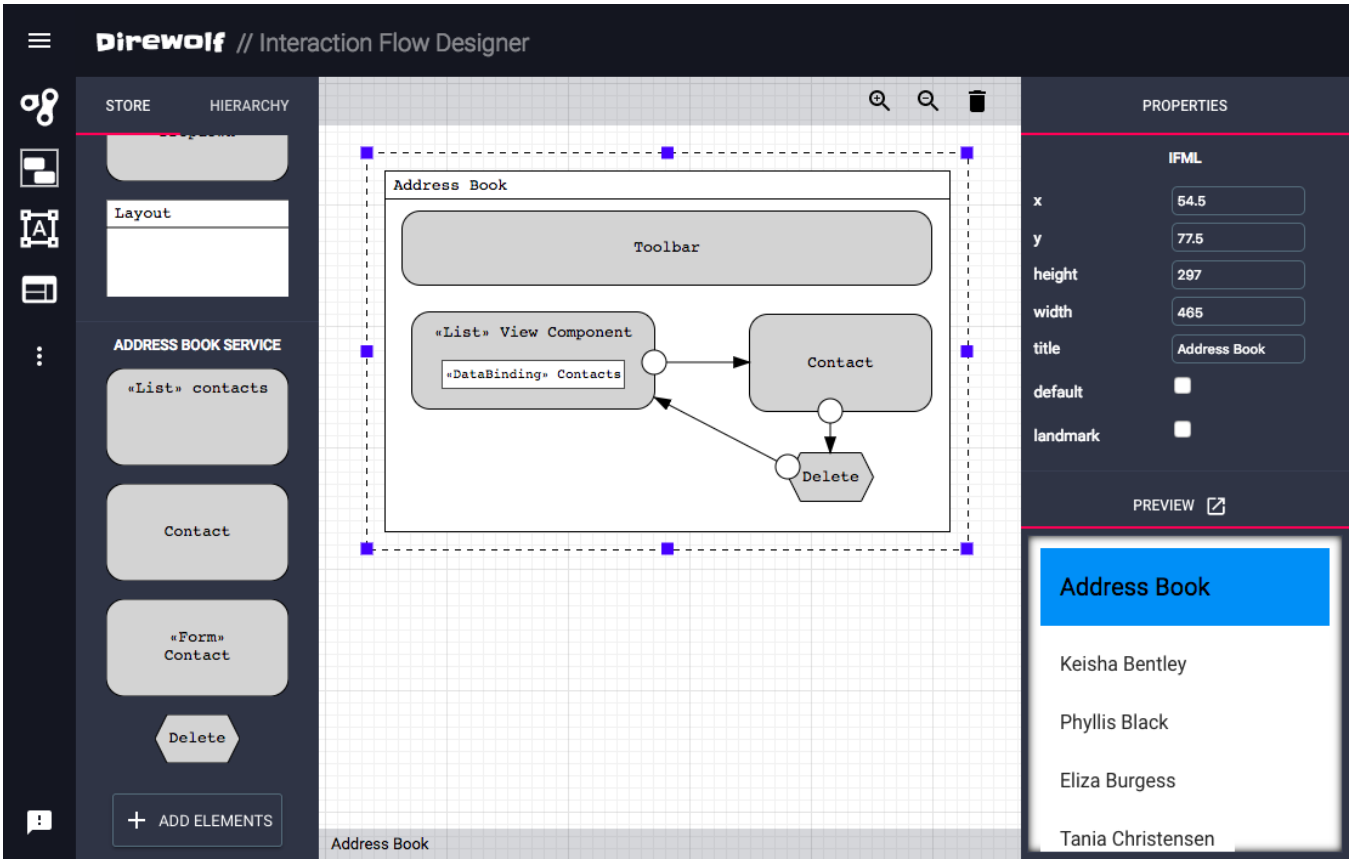

Figure 4: Interaction Flow Designer and HTML Preview

JavaScript based on a Commutative Replicated Data Types (CRDT) algorithm [18]. It is able to exchange synchronization messages over various protocols like XMPP, WebRTC and the InterPlanetary File System [2]. We employ a WebSocket solution, where a central entity on a server forwards messages to all connected peers. The underlying collaboration architecture is based on *spaces*. A space is a collection of synchronized nodes, similar to the hierarchical structure of the document object model (DOM). Each node within a space has access to a globally synchronized data store; the store is synchronized across application instances running on distinct browsers. Additionally, each instance of a node gets access to a node-specific shared data store. Nodes and edges are saved as JavaScript objects. Model transformers, e.g. the one responsible for the generation of HTML5 from IFML models, directly work on the synchronized data store. The same principle applies to the properties sidepanel on the right; it works on the shared data structures: changes are propagated to the model editor via the synchronized JavaScript object and then applied.

5.5 End-User-Oriented Frontend Designer

We also implemented a HTML5 editor as an end-user-oriented view. The visual structure of the editor follows the Interaction Flow Designer, but instead of IFML model elements, in the palette, it contains actual HTML widgets like a toolbar, an image element and a button. The HTML user interface in the center is updated in parallel with the underlying IFML model. HTML widgets can be selected; their shared properties are then shown in the property browser on the right. In the background, the HTML editor works on the shared IFML model, i.e. when an HTML widget is dragged & dropped to the HTML preview, in the background a respective IFML View Container or View Component is added to the model which then triggers the transformation into HTML.

6 DISCUSSION

After presenting the conceptual architecture and our implementation, we discuss the strengths and weaknesses of our approach in this section.

The greatest strength of our approach is the strong focus on open standards. This applies to all layers starting from the OpenAPI specification governed by the OpenAPI Initiative and the Interaction Flow Modeling Language of the Object Management Group, to the HTML5 and SVG frontend standards by the W3C. Employing Material Design guidelines helps us in accessibility aspects, as well as in targeting various device form factors by following responsive design best practices.

One of the key reasons for employing models in software engineering is the abstraction of concrete interfaces, designs and other

implementation aspects. In this regard, the lack of contextual parameters that are abstracted away does also apply to IFML. For instance, some concepts like the obfuscation of password input fields cannot be automatically derived from an OpenAPI documentation. In this particular case, manual adjustments are essential, which can be performed in the HTML editor view of our tool. Another major drawback is that currently, security parameters like an OAuth access token described within the OpenAPI specification are not handled by our tool. We are confident to be able to tackle these technological drawbacks by extending the prototype in the future. On a more general level, automation may lead to monotonous stencil-type Web applications where the craft of designers is no longer demanded. However, on the contrary, we believe that it is precisely by eliminating monotonous works that the creativity of designers can be fostered. On the long run, overcoming development aspects may trigger the end-user-driven creation of a huge variety of personal applications [14], where developers focus on delivering extensively documented APIs.

7 CONCLUSION

In this article we presented a tool for generating Web user interfaces with the help of API documentation. We therefore conceptualized a two-step process that first transforms OpenAPI documentation to an intermediary IFML model; the second step then generates HTML5 & JavaScript frontends. The implementation resulted in a collaborative Web tool built on open standards like SVG. It provides an Interaction Flow Designer as well as a HTML editor for style-based adaptation of the generated frontends. The OpenAPI documentation format is increasingly popular in the realm of REST API developers. Our tool helps automating the development of Web application prototypes, so designers and even end users can abstract the implementation details away.

Code generation from OpenAPI documents is helpful in other areas as well; we are currently investigating its usage in the visualization of data provided by REST APIs. Future work includes a component registry based on JSON schema matches that can help build an ecosystem of available frontend components. Based on the property schemes of the API, designers can download compatible elements from a repository to embed them in their Web apps. To make our tool more useful in real-world contexts, we plan to add user management and awareness functionalities, and the introduction of *Progressive Web Application* concepts via a Service Worker responsible for cashing generated Web frontends.

We strongly believe that automation is one of the keys to tackle challenges in the creation of situational applications for the long tail [1]. Particularly, we are interested in application cases in the Internet of Things. Beyond the use cases, we are researching further means of closing the gaps between developers and end users. An extensive end user evaluation will contribute to these intentions. Finally, the peer-to-peer background of our work via the Yjs library is an ideal companion to existing distributed Web initiatives like the InterPlanetary File System (IPFS) [2] and Solid [15].

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 Programme through the project "WEKIT" (grant no. 687669).

REFERENCES

[1] Chris Anderson. 2006. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion, New York.
[2] Juan Benet. 14.07.2014. IPFS - Content Addressed, Versioned, P2P File System. (14.07.2014). https://arxiv.org/abs/1407.3561
[3] Carlo Bernaschina, Sara Comai, and Piero Fraternali. 2017. IFMLEdit.org: Model Driven Rapid Prototyping of Mobile Apps. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 207–208. https://doi.org/10.1109/MOBILESoft.2017.15
[4] Marco Brambilla and Piero Fraternali. 2014. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML.* Morgan Kaufmann.
[5] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9
[6] JS Foundation. 2018. Node-RED. (2018). https://nodered.org/ [Online; accessed February 28, 2018].
[7] Google. 2018. Material Design Guidelines. (2018). https://material.io/guidelines/ [Online; accessed February 28, 2018].
[8] Jiang He and I-Ling Yen. 2007. Adaptive User Interface Generation for Web Services. In *IEEE International Conference on e-Business Engineering (ICEBE'07)*. 536–539. https://doi.org/10.1109/ICEBE.2007.82
[9] IFTTT Inc. 2018. IFTTT. (2018). https://ifttt.com/ [Online; accessed February 28, 2018].
[10] OpenAPI Initiative. 2018. The OpenAPI Specification. (2018). https://www.openapis.org/ [Online; accessed February 28, 2018].
[11] Kevin Jahns. 2018. Yjs. (2018). http://y-js.org/ [Online; accessed February 28, 2018].
[12] Farzad Khodadadi, Amir Vahid Dastjerdi, and Rajkumar Buyya. 2015. Simurgh: A framework for effective discovery, programming, and integration of services exposed in IoT. In *2015 International Conference on Recent Advances in Internet of Things (RIoT)*. 1–6. https://doi.org/10.1109/RIOT.2015.7104910
[13] Giuseppe La Torre, Salvatore Monteleone, Marco Cavallo, Valeria DAmico, and Vincenzo Catania. 2016. A Context-Aware Solution to Improve Web Service Discovery and User-Service Interaction. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. 180–187. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0047
[14] Henry Lieberman, Fabio Paternò, and Volker Wulf (Eds.). 2006. *End User Development.* Human-Computer Interaction Series, Vol. 9. Springer, Dordrecht.
[15] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*, Jacqueline Bourdeau, Jim A. Hendler, Roger Nkambou Nkambou, Ian Horrocks, and Ben Y. Zhao (Eds.). 223–226. https://doi.org/10.1145/2872518.2890529
[16] E. Michael Maximilien, Hernan Wilkinson, Nirmit Desai, and Stefan Tai. 2007. A Domain-Specific Language for Web APIs and Services Mashups. In *Service-Oriented Computing – ICSOC*, Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan (Eds.). Lecture Notes in Computer Science, Vol. 4749. Springer Berlin Heidelberg, 13–26. https://doi.org/10.1007/978-3-540-74974-5{_}2
[17] Tobias Nestler, Marius Feldmann, Gerald Hübsch, André Preußner, and Uwe Jugel. 2010. The ServFace Builder - A WYSIWYG Approach for Building Service-Based Applications. In *Web Engineering*, Boualem Benatallah, Fabio Casati, Gerti Kappel, and Gustavo Rossi (Eds.). Lecture Notes in Computer Science, Vol. 6189. Springer, Berlin, Heidelberg, 498–501. https://doi.org/10.1007/978-3-642-13911-6_37
[18] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. 2016. Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In *GROUP 2016*. ACM. https://doi.org/10.1145/2957276.2957310
[19] Object Management Group. 2015. Interaction Flow Modeling Language. (2015). http://www.omg.org/spec/IFML/
[20] SmartBear Software. 2018. Swagger UI. (2018). https://swagger.io/swagger-ui/ [Online; accessed February 28, 2018].
[21] Mandana Vaziri, Louis Mandel, Avraham Shinnar, Jérôme Siméon, and Martin Hirzel. 2017. Generating Chat Bots from Web API Specifications. In *The 2017 ACM SIGPLAN International Symposium*, Emina Torlak, Tijs van der Storm, and Robert Biddle (Eds.). 44–57. https://doi.org/10.1145/3133850.3133864