Quid: Prototyping Web Components on the Web

Pedro J. Molina
pjmolina@metadev.pro
Metadev S.L.
Sevilla, Spain

ABSTRACT

Quid is a web modeling tool aiming to create a complete modeling environment for prototyping **Web Components** in the browser. The approach presented introduces a concise DSL based on minimal syntax and indentation to implement containment relationships. A WYSIWYG editor is also provided to allow users to explore in real-time the design been constructed. Model-driven techniques and code generation are used to explore different implementation choices with different Web Components frameworks.

CCS CONCEPTS

• **Information systems** → **Browsers**; • **Software and its engineering** → **Interpreters**; **Domain specific languages**.

KEYWORDS

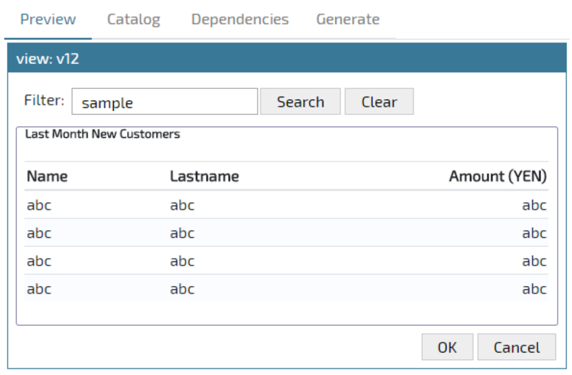
web components, domain specific languages, web, WYSIWYG, prototype, user interface, modeling on the web, code generation

ACM Reference Format:
Pedro J. Molina. 2019. Quid: Prototyping Web Components on the Web. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '19), June 18–21, 2019, Valencia, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3319499.3330294

1 INTRODUCTION

User Interface construction is an omnipresent topic in Software Engineering: multiples tools ranging paradigms such us textual, graphical design tools are available to help in this task. At the same time, the fast pace of the evolution of front-end industrial frameworks makes such editors tools obsolete as soon as new technology emerges.

The work presented (**Quid**) introduces a web-based *Domain Specific Language* with a strong focus on *minimal accidental complexity*, removing the accessory markup and a WYSIWYG (*What You See Is What You Get*) environment to provide real-time feedback to users (see Figure 1). Moreover, the User Interface specification built in this way is platform-independent: its primitives can be extended, and target different platforms: using model transformations and code generation for generating software artifacts like Native Web Components or Angular code.



Name	Lastname	Amount (YEN)
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figure 1: Quid Preview pane

2 MOTIVATION

The rapid evolution of the web technologies, the increasing accidental complexity found on tooling and platforms and the fragmentation of devices can make Web development a daunting task. Let's review briefly how web technologies have evolved.

2.1 Evolution of User Interface Paradigms on the Web

In 1990 Tim Berns-Lee[2] invented the WWW. The first version introduced hypermedia concepts and basic rendering oriented to sharing information in the form of linked documents. **Web 1.0** was called *read-only web* in the sense all content was static to be consumed as it is published. JavaScript as the first scripting language to be supported in-browser was introduced in 1996[19] and AJAX as the way to invoke asynchronously calls to server-side emerged in 1999.

These changes enabled **Web 2.0** circa 2004, the *read-write web* allowed crowd of people with common interests to collaborate emerging the first social networks. Technologies like CGI, Perl and PHP allowed the creation of web applications with dynamic content server-side generated.

Web 3.0 focused on semantic web introducing machine-readable formats to enable the automation of tasks and information retrieval.

Recently, more and more presentation logic has been moved to the browser to increase application's scalability and to take an advantage of the increased computing resources in the browser especially to run JavaScript in a very optimized way arriving into the era of SPA (*Single Page Applications*) fueled by different JavaScript frameworks like React, Angular, VueJS, Aurelia, Elm or CycleJS.

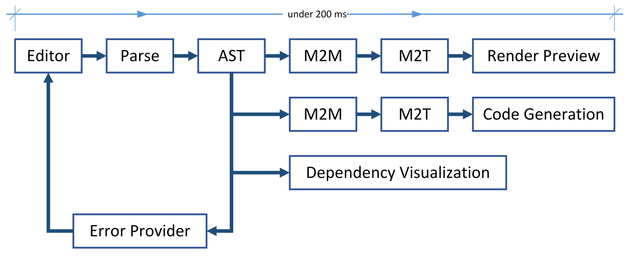


Figure 2: Quid Architecture

These frameworks follow different approaches to develop User Interfaces: from traditional Model-View-Controller architectures, passing from reactive and also component-based architectures.

2.2 Web Components

The W3C's **Web Component** specification[3] is a live specification standardized and already well implemented and supported in industrial browsers.

The Web Component approach simplifies the application development, in the way it brings a full a Component Model for user interfaces in a native way (implemented by browsers directly and exposing a common API to developers). This allows the mix of components developed with different languages, or frameworks to compose a web page or application and at the same time has set sensible expectations for inter-operation and the possibility of adding new components as the web evolves.

2.3 The need for tooling

Web Components technology is ready to use on desktop and mobile browsers. However, at the time of writing this, there is still a **lack of tooling for composing Web Components**. **Quid** is a tool to help in the composition of Web Components.

3 QUID

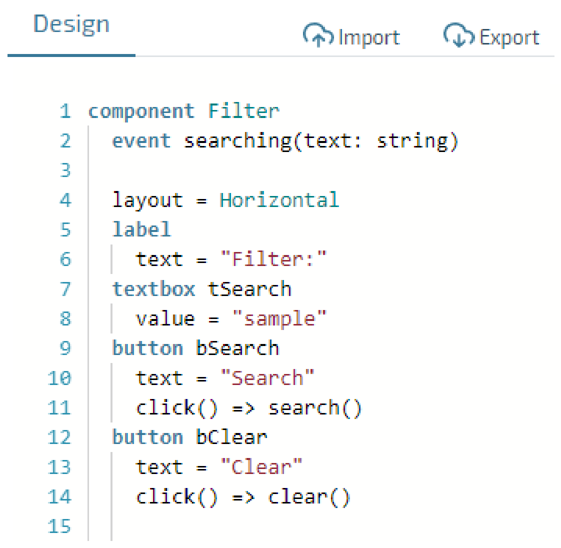
3.1 Domain-Specific Languages for the Web

Domain-Specific Languages *(DSL)* are an excellent tool to narrow the scope of a problem like User Interface Prototyping. However, the creation and usage of complete DSLs traditionally used to require a lot of heavy tooling like EMF on top of Eclipse, or Microsoft DSL Tools on top of Visual Studio. The Web platform nowadays provides full capabilities to create rich applications in-browser without the need for the user to install local software and update it regularly. Commercial DSL tool creators like Obeo, TypeFox, Itemis, and Metadev and exploring and developing web-enabled technologies to allow creating and using DSLs on the web.

Quid is a web tool exposing a simple DSL to describe User Interfaces. A general overview can be seen in Figure 3. In the following sections, the main features will be described.

3.2 Foundations

Quid uses a component model of AIOs (*Abstract Interaction Objects*) as defined by [1]. As a component model exposes controls as classes: instantiated objects with properties, events, and methods. A Quid specification is a textual description of a tree of AIOs



```
component Filter
  event searching(text: string)

  layout = Horizontal
  label
    text = "Filter:"
  textbox tSearch
    value = "sample"
  button bSearch
    text = "Search"
    click() => search()
  button bClear
    text = "Clear"
    click() => clear()

```

Figure 3: Quid Editor Pane

where properties are bind to values and expressions, and events are registered with listeners to trigger actions on response. Moreover, the composition of AIOs produces and expose, in a recurrent manner, patterns that can be captured and reused as *Conceptual User Interface Patterns*[17].

Figure 2 shows the general architecture of the tool. The editors handles all the text editing task including colorization, and code completion. Input text is parsed using a PEG parser in real time (parsed on every user click) and generates an AST (*Abstract Syntax Tree*) in memory. A validation process is executed using the AST to check for errors. In case of errors (at parsing level: syntactic or AST level: semantic) they are reported inline directly to the editor as close as possible to the error source with a human-friendly error message.

On the other hand, if no critical errors are found, the AST is transformed via a M2M (*Model to Model*) transformation and later on via a M2T (*Model to Text*) transformation to produce plain HTML, CSS3 and JavaScript to provide a render preview. All these steps from parsing to preview are done under 200 milliseconds to provide **immediate feedback** to the user.

On demand, the model can used for code generation to produce the User Interface in different target frameworks for Web Components. The process here uses a similar composition of M2M and M2T transformations to produce the specific code for such framework.

Additionally, the Dependency Graph is generated transformed the model in a similar way into a dependency graph data for a visualization library.

3.3 DSL

Quid provides a textual DSL with minimal markup (brackets and delimiters) to avoid unneeded accidental complexity. Indentation is used to represent **visual containment** relationships. This means interaction objects can be nested just using the TAB key and reversed using Shift-TAB. The DSL editor is text-based providing colorization and code completion using the key shortcut Control-Space.

Sample UI specification on Quid:

```
component Filter
    label l1
       text = "Filter:"
    textbox searchText
       tooltip = "type here for search"
    button cmdSearch
     text = "Search"
       (click) => onSearch()

view Page1
    Filter f1
    ...
```

In the sample provided, a component (or AIO) named **Filter** is defined. The indentation suggests the next three components are defined inside the Filter. This containment relationship is both graphical, and semantic (scope and encapsulation). The sub-components have properties and can be configured as needed. Finally, an event is captured to trigger an execution after a **click** event is produced on a button. The component defined in this way is immediately available for use in the view **Page1** defined below, where a **Filter** is instantiated with the name of **f1**.

Components in Quid are AIOs in the sense they abstract interaction components and can be rendered later to CIOs (*Concrete Interaction Objects*) when transformed to concrete implementation for an specific target platform.

3.4 Tool Support

Integrated Validation. The DSL provides a real-time integrated validation. On each keystroke, a validation procedure is triggered involving: tokenization and parsing stage, AST (*abstract syntax tree*) construction, semantic validation, and error reporting. If lexical or semantic errors are found, they are provided just in time in the zone below the editor and inline in the editor using colors and decorators to indicate the source of the errors in the exacts lines.

3.4.1 Preview Mode. Once the validation is successful, the tool provides a *Preview window* where the actual design is rendered to provide a WYSIWYG experience with immediate feedback to the designer. In this kind of tools, this immediate feedback is the key to enhancing and keeping the productivity of users within the tool. The preview mode allows interaction, depending on the behavior defined for the User Interface.

3.4.2 Component Catalog. Quid is extensible in terms of basic building blocks to be used for design. The catalog (see Figure 4) is the place to explore the available primitives, its properties and events, and to import third-party components when needed. Changing the base building blocks allows Quid to be used in different design scenarios without changing the core tool. Moreover, third-party components can be incorporated to change the palette of building blocks.

3.4.3 Dependency Tree. A *Dependency Tree* pane (see Figure 5) in Quid is provided to explore visually the following relationships:

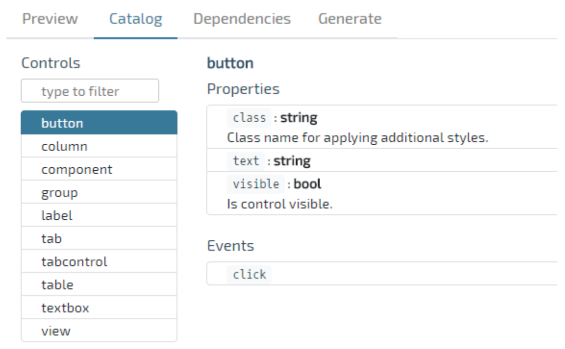


Figure 4: Quid: Component Catalog pane

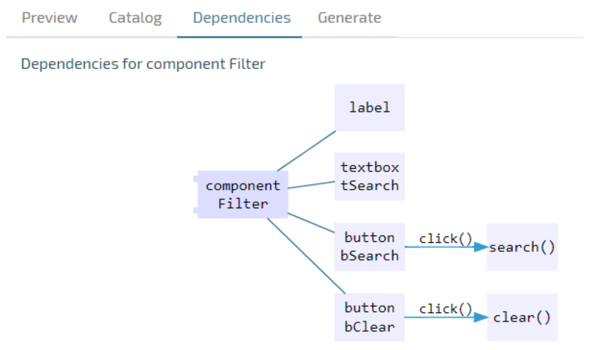


Figure 5: Quid: Dependency Tree pane

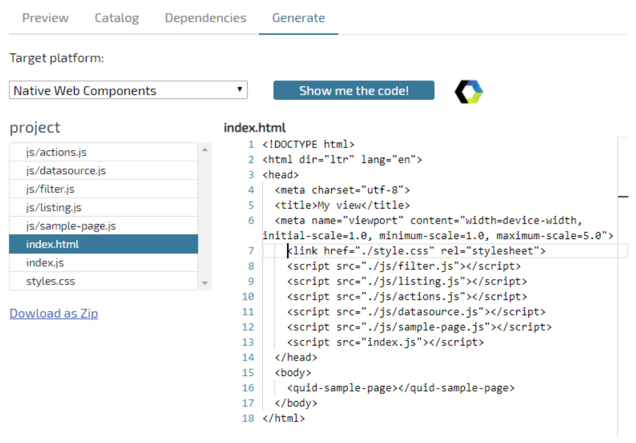
- AIO containment: parent-child relations that can be recursive.
- Event subscription and event handlers.

3.4.4 Code Generation. Finally, a code generation pane (see Figure 6) provides an integrated tool-set to translate the specification into different code artifacts depending on the target platform. **Quid** provides code generation to Native Web Components generating HTML, CSS & JavaScript code to allow a running implementation of the spec. Currently, the following technologies are supported for code generation: Native Web Components (with or without a SystemJS loader), Angular Elements, Polymer 3 and Stencil. Others frameworks under consideration for inclusion are LitHtml, VueJS, Aurelia, and React.

4 USE CASES

The tool supports the following use cases:

(1) Define components and composition of components for fast prototyping of user interfaces (Atomic Design).
(2) Form design.
(3) Selection of a closed catalog of components (company palette) to enforce common look & feel on designed user interfaces.
(4) Custom code generation for arbitrary architectures and language stack.
(5) Use the Quid interpreter to render the user interfaces on third-party applications.



```
<!DOCTYPE html>
<html dir="ltr" lang="en">
<head>
  <meta charset="utf-8">
  <title>My view</title>
  <meta name="viewport" content="width=device-width,
initial-scale=1.0, minimum-scale=1.0, maximum-scale=5.0">
    <link href="./style.css" rel="stylesheet">
    <script src="./js/filter.js"></script>
    <script src="./js/listing.js"></script>
    <script src="./js/actions.js"></script>
    <script src="./js/datasource.js"></script>
    <script src="./js/sample-page.js"></script>
    <script src="index.js"></script>
  </head>
  <body>
    <quid-sample-page></quid-sample-page>
  </body>
</html>
```

Figure 6: Quid: Code Generation pane

4.1 Industrial Experiences

The tool has been used with success during 2018 in an international project to design and generate User Interfaces for an Investment Banking Application led by the international software consultancy firm **Everis**.

The project used a dot.NET Core architecture for the back-end services exposed by OpenAPI contracts plus an ASP.NET MVC front-end. Quid was added as a Web DSL for capturing the User Interface requirements using a grammar implemented with PEG.js[4] with a TypeScript target TS-PEG.js[15]. It was used in combination with Everis's internal project **iwok** for automation. Custom code-generation were written to target the architecture and specific UI controls used in the project.

Project results:

(1) Automation and code-generation saved an estimation of 1.900 hours of development (scaffolding, prototype, and testing).
(2) Estimation of 25% less effort in the construction and unit-testing for new developments (new modules).

4.2 Related works

Modeling on the Web related projects include: **Microsoft Windows Phone App Studio**[6] (2013), **Buildup.io**[7] (2014) and **IBM Bluemix Mobile App Builder**[9] (2015) both oriented to create fully native mobile applications for non-developers. The tools provided a web-based SPA WYSIWYG editor for non-experts to build content and basic navigation using drag&drop and a palette of components. A cloud-based engine was able to transform the model into a native application by means of different levels of M2M and M2T code generation plus steps for adding assets, compilation, packaging and direct download to the user's phone using BIDI codes. The former one targeting Windows Phone applications in C# and the latter ones Android apps with Java and iOS apps with Objective C, respectively.

DSL Forge[12] (2014) is an open source project able to publish a XText or ANTRL based DSL on the web.

Hivepod.io[8] (2015) is another web tool exploring textual and projectional edition of a structural model. Such model is used generate a full backend using a MEAN stack (MongoDB, Express, Angular and NodeJS) and with capabilities to be deployed in cloud providers like Heroku, IBM Bluemix, Digital Ocean, Cloudshare and others.

Theia IDE[20] (2017) from TypeFox builds on top of the **Monaco Editor**[14] from Microsoft (the same base editor used in Quid) to build a complete Web IDE capable to expose textual using XText as a base and graphical DSLs.

Obeo is working on bringing **Sirius** to the Web also[18] (2018). Sirius provides diagrams and graphical editors traditionally build inside the Eclipse IDE. Now building with SVG technology similar technology for the web.

Convecton [10] (2018) is a proposal and work in progress from Itemis to expose DSL on the web using MPS[11] as an underlying server-side backend.

5 CONCLUSIONS & FUTURE WORKS

5.1 Conclusions

Although there are myriad of tools for designing User Interfaces, tools for designing **Web Component** are scarce. The main novelty of **Quid** is to focus on:

(1) abstract UI representation,
(2) providing a web-based DSL (no installation of software is required, bringing a full-featured editor on the web), and
(3) early embracing a component model fully compatible with W3C Web Components to help using, importing and combining such web components.

The *Immediate Feedback* principle is also a great feature exposed by the tool helping users to understand immediately how changes affect the design. This feedback could be slowed as the specification grows. To mitigate such kind problems, techniques are been applied to partition the specification in separate chunks and to render partial views the **Preview** pane.

5.2 Future works

One of the features still not standardized in Web Components is a **format for metadata interchange** exposing and documenting properties, events, methods, parameters with its corresponding types[16]. The base language used for Web Components, JavaScript, is in its nature a dynamic type language. On the other hand, languages like TypeScript[5] introduces progressive type-safety that helps to define contracts and to ensure are used property. Web Components define a contract when used by a consumer, that's why it is vital to enforce such type information on Web Components. This standardization will definitely contribute to creating an ecosystem of tools for creating and consuming Web Components.

The latest version of **Quid** can be found online and ready to use for free at: https://quid.metadev.pro[13].

ACKNOWLEDGMENTS

Dedicated with love, to my first daughter Lucia and her mother Nuria, to be born on mid-May of 2019.

REFERENCES

[1] François Bodart and Jean Vanderdonckt (Eds.). 1996. *Design, Specification and Verification of Interactive Systems'96, Proceedings of the Third International Eurographics Workshop, June 5-7, 1996, Namur, Belgium.* Springer.

[2] World Wide Web Consortium. 1990. The World Wide Web Consortium, 1990. Retrieved April, 2019 from http://info.cern.ch/hypertext/WWW/TheProject.html

[3] World Wide Web Consortium. 2018. The World Wide Web Consortium, et al. Web Component Specification Working Drafts, 2014-2018. Retrieved April, 2019 from https://www.w3.org/standards/techs/components

[4] David Majda et al. 2010. PEG.js. Retrieved April, 2019 from https://pegjs.org

[5] Anders Hejlsberg. 2012. Introducing TypeScript. *Microsoft Channel* 9 (2012).

[6] Icinetic. 2013. Create your own app with Windows Phone App Studio, Microsoft. Retrieved April, 2019 from https://blogs.windows.com/devices/2013/08/20/create-your-own-app-with-windows-phone-app-studio

[7] Icinetic. 2014. Buildup.io. Retrieved April, 2019 from https://buildup.io

[8] Icinetic. 2015. Hivepod. Retrieved April, 2019 from https://hivepod.io

[9] Icinetic. 2016. Build stunning mobile apps with IBM Bluemix mobile services. Retrieved April, 2019 from https://www.youtube.com/watch?v=U1y4r_t4L2g

[10] Itemis. 2018. Convencton (work in progress). Retrieved April, 2019 from https://www.itemis.com/en/convecton

[11] Jetbrains. 2008. Meta Programming System. Retrieved April, 2019 from https://jetbrains.com/mps

[12] Amine Lajmi, Jabier Martinez, and Tewfik Ziadi. 2014. DSLFORGE: Textual Modeling on the Web. In *Proceedings of the Demonstrations Track of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014),Valencia, Spain , October 1st and 2nd, 2014. (CEUR Workshop Proceedings)*, Tao Yue and Benoît Combemale (Eds.), Vol. 1255. CEUR-WS.org, Valencia, Spain. http://ceur-ws.org/Vol-1255/paper5.pdf

[13] Metadev. 2018. Quid. Retrieved April, 2019 from https://quid.metadev.pro

[14] Microsoft. 2016. Monaco Editor. Retrieved April, 2019 from https://microsoft.github.io/monaco-editor

[15] Pedro J. Molina. 2017. TS PEG.js. Retrieved April, 2019 from https://www.npmjs.com/package/ts-pegjs

[16] Pedro J. Molina. 2018. Quid: a tool for composing Web Components. Retrieved April, 2019 from https://medium.com/@pjmolina/quid-6d89b0ec58d5

[17] Pedro J. Molina, Oscar Pastor, Sofía Martí, Juan Jose Fons, and Emilio Insfrán. 2001. Specifying Conceptual Interface Patterns in an Object-Oriented Method with Automatic Code Generation. In *UIDIS*. IEEE Computer Society, Zurich, Switzerland, 72–79. https://doi.org/10.1109/UIDIS.2001.929927

[18] Obeo. 2018. Modeling tools go up to the cloud. Retrieved April, 2019 from http://melb.enix.org/2018/02/22/up-to-the-cloud

[19] Google Chrome Team. 2010. Evolution of the Web. (interactive resource). Retrieved April, 2019 from http://www.evolutionoftheweb.com

[20] TypeFox. 2018. Theia IDE. Retrieved April, 2019 from https://www.theia-ide.org