Rapido: A Sketching Tool for Web API Designers

Ronnie Mitra
Director of API Design
CA Technologies
ronnie.mitra@gmail.com

ABSTRACT
Well-designed Web APIs must provide high levels of usability and must "get it right" on the first release. One strategy for accomplishing this feat is to identify usability issues early in the design process before a public release.

Sketching is a useful way of improving the user experience early in the design phase. Designers can create many sketches and learn from them.

The Rapido tool is designed to automate the Web API sketching process and help designers improve usability in an iterative fashion.

Keywords
API, design, sketching, REST, Web API, usability

1. INTRODUCTION
Over the years, a number of books, papers and talks have been published to provide design guidance for Web API Designers.

The expert advice in this domain often highlights three important characteristics for well-designed APIs:

- APIs should provide 'just enough' functionality to meet user requirements
- APIs should follow a user-centered design approach
- APIs should not break existing applications

This advice presents an interesting challenge for the modern API designer. They must create interfaces that provide good user experiences within the constraint of limiting breaking changes.

Joshua Bloch illustrates this challenge in one of his maxims for good design: "Public APIs, like diamonds, are forever. You have one chance to get it right so give it your best." [1]

This paper introduces sketching as a method for improving API user experiences in the design phase, before implementation. It also introduces a tool called Rapido that has been created to facilitate the act of sketching web APIs.

2. SKETCHING
An implicit but important part of many design processes is the sketching phase. Sketching has been proposed as a formal component of many design processes including interaction design [2], mechanical design [3] and engineering[4].

In broad terms, a sketch is a form of design that includes relatively less detail than a formal design. Sketches carry connotations of being formed quickly and effortlessly.

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2743040

They often capture abstract thoughts from a designer's mind and can be used to record experimentation with new ideas that might otherwise be forgotten.

In his book "Sketching User Experiences", Bill Buxton highlights the importance of sketching in all design processes and identifies that sketching is central to design thinking and learning[5]. Buxton also explains how sketches can be used to improve the user experience within an interaction design context by allowing designers to produce many experimental concepts that are iteratively refined in pursuit of the best final concept.

For the Web API designer, sketching offers a method to achieve rapid design iterations at the beginning of a design process. By creating a sketch, reviewing it and applying the lessons learned to a new sketch a designer can identify and solve usability problems in the early phases of design.

When incorporated into a design process that includes user testing and prototype development, the designer can improve the odds that the overall user experience will be improved.

There are many ways to produce sketches, but we will focus on four popular sketching methods and discuss the benefits and limitations of each.

2.1 Drawing
A simple and accessible method for sketching is to put pencil to paper, marker to whiteboard or digital pen to canvas.

Drawing can be a powerful way of representing the memories or abstractions of the human mind. While the quality level of a drawing may differ depending on the artist's level of artistic skill, the use of a drawing to communicate an idea is as central to the human experience as spoken language[6].

2.1.1 Benefits
One of the great advantages to drawing a sketch is the ease with which one can start.

Almost every designer should be capable of effortlessly creating a doodle. This simple act doesn't require strenuous thought regarding the process of drawing. But, these rough sketches can provide a sufficient level detail to capture the important aspects of an early phase design regardless of their aesthetic appeal.

The human proficiency for drawing simple images and the speed at which a low fidelity drawing can be rendered makes it easy for the sketch artist to draw many sketches in a short time, disposing of sketches as they go.

For example, a designer considering a CRUD API might initially draw the low fidelity sketch depicted in figure 1 using simple boxes and lines.

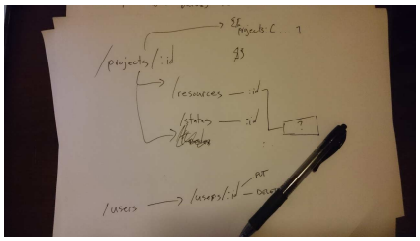

Figure 1 - Drawing a CRUD API Sketch

Another of the drawing method's great strengths is that it fosters experimentation within the sketching process. Designers that start with a blank canvas are free to experiment with new ideas and concepts in a limitless fashion, bounded only by the physical or digital constraints of the specific medium they choose.

For example, a designer might sketch interfaces with new features that do not conform to existing standards, or interfaces that defy an existing understanding of system and implementation limits.

The freedom to innovate and capture new ideas and concepts with little effort are powerful features of the drawn sketch.

2.1.2 Limitations

Unless the designer is using a specific drawing tool, drawn sketches are ill-suited for API implementation. For example, it is difficult to introduce enough fidelity into the sketch for a useful implementation and the medium itself cannot easily be translated to machine code.

In addition, the freedom to draw and experiment on a page can be as limiting as it is beneficial. The blank page provides no guidance for designers who have little experience in API design. The designer who draws a sketch must understand the basic rules that characterize their chosen API style in order to draw an effective interface sketch.

Capturing a high level of detail can also be difficult when drawing a sketch. While, drawing is well-suited to capturing the important abstract entities and relationships of an interface, it is difficult to capture higher fidelity information. For example, listing operations, parameters and message bodies for an interface can easily become a labour intensive exercise without some form of tooling.

2.2 Writing

While the drawn sketch is a visual representation of the designer's abstract idea, a 'written' sketch is a literal representation.

The actual language used for the sketch can take many forms. For instance, a designer familiar with the Ruby programming language might sketch their HTTP based API in Ruby code. The same designer might also sketch their API is pseudo-code when describing it in an email on a mailing list.

2.2.1 Benefits

As with drawing, writing is a common method for humans to communicate and store abstract ideas. We are inherently driven to describe abstract concepts either visually or literally [6], so the act of writing out early designs of the API is a natural and easy sketching method to adopt.

Written sketches become particularly powerful when programming languages are used. Designers not only get the benefit a sketch that can be communicated in a common, standardized language, but also gain an API that can be invoked. This can greatly increase the testability and socialization of the interface as well as reduce the prototype implementation time.

Writing or coding an API is a natural way to build powerful prototypes and can be an effective option for sketching as well.

2.2.2 Limitations

Although writing an API in a programming language can improve its implementability, designers who take this approach may quickly find themselves focusing on very detailed implementation decisions rather than the abstract design decisions that should be the focus in a sketching phase.

In addition, when sketching an API design in a programming language, designers can become focused on solving the problems of correct syntax, error handling and code readability that are important for implementation instead of the abstract design problems related to the interface.

Writing an API as a first step can also present disposability problems. Designers may grow emotionally attached to the code they have crafted and they may re-use code between sketch iterations rather than starting over with a new idea.

Even worse, if the programming language used to sketch the API is the same language that will be used for implementation, the line between sketch, prototype and build is easily blurred. Maintaining a low-fidelity, disposable sketch in this type of circumstance is very difficult.

2.3 UML

The UML (Unified Modeling Language) is an existing and widely used standard for modeling object oriented software systems. The UML addresses five views of software architecture and can be used to model abstract systems like Web APIs [9].

When used for sketching, UML offers a specialized form of both the drawing and writing methods.

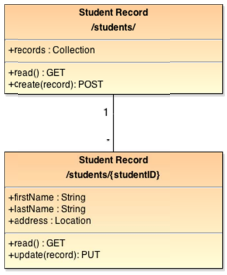

Figure 2 UML Diagram for an object based API

2.3.1 Benefits

A great feature of UML is that it is formally defined and offers a predictable syntax. This modeling standardization has made it possible for UML based tooling to emerge aiding the process of UML creation. In addition, some UML tools can auto-generate application code based on the model, improving the implementability of a UML based sketch.

Another advantage of UML is that it does not prescribe how an API should be designed. In fact, any network-based architecture should be able to be depicted with this sketching method, providing a high level of experimentability for designers.

2.3.2 Limitations

However, UML's flexibility also presents a usability challenge for API designers[7]. The modeling language is designed for general use and not for the specific domain of Web API design.

For example, URLs, query parameters, HTTP nomenclature, media types and vocabularies must all be mapped to the class, property and method taxonomy of a UML class diagram. Designers must gain enough expertise with the UML language to be able to apply it to their own domain problems.

One solution to this problem might be to implement a domain specific modeling language that uses UML. For example, the Object Constraint Language (OCL) was originally an extension of UML designed for object oriented analysis and design[10]. However, as of this writing a popular UML based Web API language has not emerged.

In addition, the syntax of UML presents a socialization problem. A UML based sketch cannot be shared with others unless they also possess the appropriate level of UML knowledge to understand it, limiting the appeal of UML as a sketching method.

A designer who is competent with UML will be able to quickly translate abstract thought to a conceptual version of the API. However, learning the modeling language while designing an interface can present a difficult challenge.

Finally, some UML users use the modeling language primarily as a method for code generation or documentation of production code[7]. This perception of UML as a programming or documentation language may limit the designer's desire to use the modeling language for informal sketching.

2.4 Interface Description Editors

There is an emerging set of service description formats that attempt to describe Web and REST APIs. The list of description formats is constantly evolving, but at the time of this writing popular choices include WADL, Swagger, RAML and Blueprint [8].

Service description formats are not usually designed specifically to support sketching activities, but their ability to document and describe relevant details of an interface makes them a natural fit.

Ultimately, using a format language to sketch an API is akin to sketching in code. It shares all of the characteristics, benefits and limitations that were highlighted in section 3.2.

However, in recent years many service description formats have been released with editing tools to support them. These tools offer a high level of usability to designers and make the process of sketching an API much easier. For example, an API can be described in the Blueprint description format using a web based editor provided by the format authors [12] (See Figure 3).

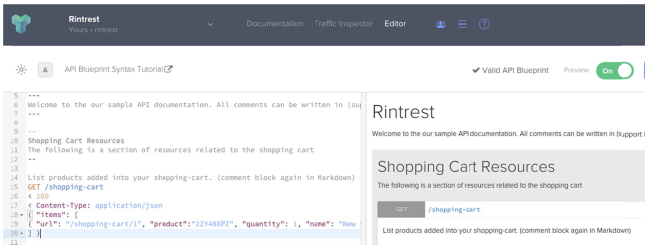

Figure 3 - Apiary.io: an editor for the Blueprint language

2.4.1 Benefits

Service description editors are powerful because they are designed to support the task of rapid interface description composition. Although designers must learn the format and syntax of the particular description language, editors with high levels of usability will facilitate that task during the design effort.

The best service description format tools provide a complete ecosystem or workflow that makes it easy to move from the sketch phase to implementation. This means that designers can easily turn their best sketches into working prototypes.

In addition, the low cost of creation combined with the ease at which text can be stored makes this method well suited for disposability. A designer can easily perform many iterations of a sketch design by starting over again

2.4.2 Limitations

Utilizing a service description format that is specifically designed to describe interfaces makes many design tasks easier, but it also makes it difficult for designers to go beyond the model that the description format author has created.

While designers can experiment with interfaces that conform to the syntax of the format in use, they are constrained by the features that the format provides. This leads to the design of interfaces that reflect the format designer's mental model.

Also, many format languages are closely associated with runtime API engines and products. This can reduce the implementability if the designer and implanter do not share the same tool ecosystem.

Finally, many interface description editors lack the holistic visual depiction of the interface model that is provided in the drawing or UML sketching methods. This lack of visualization can make it more difficult for designers and reviewers to perceive the transitions and relationships between elements of the interface.

3. RAPIDO

Rapido[11] is an experimental tool designed to facilitate web API sketching with both visual and literal methods. The goal of the experiment is to build a tool that helps designers create sketches quickly and with just enough detail to capture important API design features.

The Rapido tool is purposefully designed to constrain the designer to a relatively low-fidelity sketch when compared to methods such as written programming languages or service description editors. For example, designers can only define static responses, cannot implement conditional or error handling and cannot edit protocol header information.

A Rapido user typically follows these steps when creating a new sketch:

1. Create a sketch project
2. Define a vocabulary
3. Sketch the API
4. Model response data
5. Export an API description

With the exception of the first project creation step, each step in the sketch process can be re-visited or skipped entirely as there are no dependencies between stages.

3.1 Project Creation

To begin the act of sketching an API in Rapido, the designer must first create a new sketch project. At this stage, the designer must decide what style of API will be sketched.

At present time, Rapido supports two API styles called "CRUD" and "Hypermedia" respectively. The "CRUD" style is useful for web based interfaces that are primarily object based, while the "Hypermedia" style allows designers to sketch APIs with hypermedia controls.

3.2 Vocabulary Definition

A unique feature of the Rapido sketching process is that designers begin by creating a vocabulary for their API. The vocabulary is made of the words that are expected to appear frequently in the interface.

For example, when sketching an API for a system of student records we might start by including words such as 'student', 'report' and 'course'.

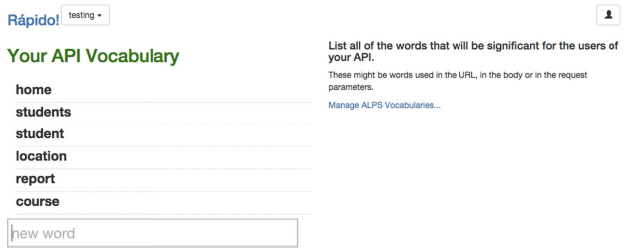

Figure 4 - Vocabulary Editor

Beginning with a list of relevant words allows the API designer to think about the interface in its most abstract terms. At this stage the designer can consider the interface based on the objects, activities and relationships that it is composed of without solving the problem of how these concepts are connected or implemented.

The words that are defined in this vocabulary stage are used in the later sketching phase automatically as type-ahead hints. This not only promotes consistency of the API's vocabulary, it also reduces the typing effort for the Rapido user.

It is important to note that the semantics of this vocabulary are not captured in the tool. The words are simply a list of strings and the sketching functions of Rapido cannot take advantage of the meanings of words in this list.

3.3 Sketching CRUD APIs

The CRUD sketching canvas reflects an object, URI and HTTP method view of a web API. This perspective of API design should resonate with most designers as these components form the building blocks of the CRUD API conceptual model.

When sketching a new CRUD style API, the designer is presented with a blank canvas and a single root node (see Figure 5). By clicking on the root node, the designer is able to create the first resource of the CRUD API.

Figure 5 – The blank CRUD canvas

A 'wizard' for resource creation is presented and the designer can enter the URI, allowed methods and query parameter details for this resource (see Figure 6). Once created, the Rapido tool provides a visual representation of the resource that includes its descriptive name, URI and allowed methods.

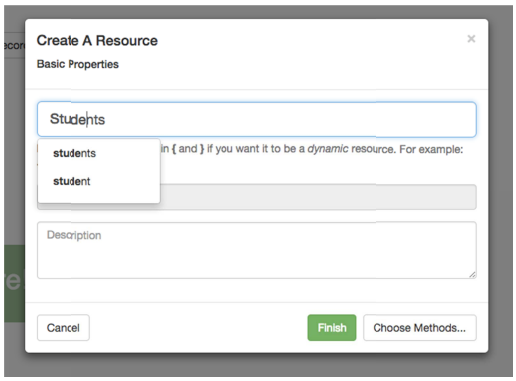

Figure 6 - Creating CRUD Objects

Designers can also compose parent-child relationships between the resources they create. The simplest way to do this in the Rapido tool is by clicking on the 'plus' icon of an existing resource. The child resource creation workflow is identical to the wizard described earlier, except for the fact that the URI field is pre-populated with the parent resource's URI. The designer is free to append to this pre-populated URI or overwrite it completely.

For example, we might sketch a collection of student records as a resource with a URI of "/students". We could also sketch a child resource of students that points to a specific student record. This child resource could have a URI of "/students/student_1".

The serialization of parent-child relationships in the URI is not a formal standard for CRUD style APIs. However, it is a regularly used convention and is recommended in many best practices guides for REST design. The URI pre-population feature is included based on an assumed popularity of this convention amongst CRUD style API designers.

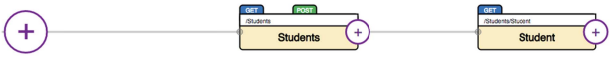

Figure 7 - Modeling object relationships

The CRUD sketching canvas is intended to let Rapido users immediately conceptualize the object and relationship model of the API. The structural elements of the API are exposed, providing an opportunity for designers to play with the API holistically and experiment with different conceptual models.

At this level of fidelity, none of the request or response message details are accessible for viewing or editing. This is by design. The goal is to discourage designers from investing too much time in a high fidelity sketch at the beginning of the design process.

3.4 Sketching Hypermedia APIs

A different type of canvas is presented to hypermedia API designers. In Rapido, a hypermedia API is sketched as an informal state diagram.

State diagrams can be useful for designing and conceptualizing hypermedia applications. Zuzak, Budiselic and Delac suggest that "transferring resource representations for transitioning agents from one state to another, suggests the usage of a state transition system formalism" [13]. Amundsen and Richardson recommend drawing a

state machine as an early step in a hypermedia API design process [14].

When a designer begins a hypermedia sketch in Rapido, they are presented with a single home state from which they may create transitions to new application states. All application states can have transitions to other application states. The goal in this canvas is to quickly sketch a web of state transitions (see Figure 8).

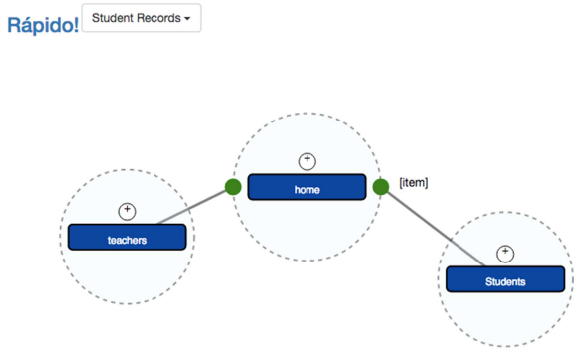

Figure 8 - Sketching a hypermedia API

The particular method for creating transitions differs depending on the media type that has been selected during project creation. Rapido currently has support for the Collection+JSON and HAL media types only, but the goal is to extend support to other types in the future.

For example, when sketching a Collection+JSON API we can create a transition to a new child item by stepping though a wizard that is specific to the Collection+JSON specification. We can choose what type of link we want, where it should be located and what the target should be (see Figure 9). In turn, the state diagram contains visual cues that are relevant to this media type. The different types of Collection+JSON links can be differentiated by colour and pattern.

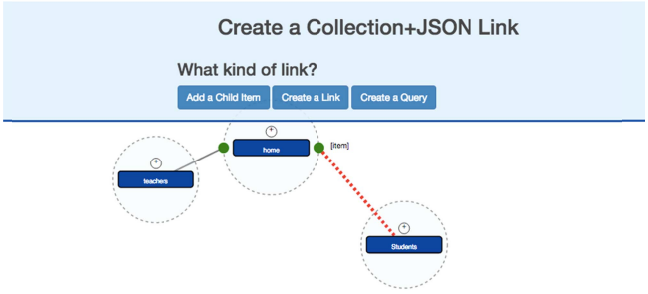

Figure 9 - Creating new Collection+JSON transitions

The fidelity of the hypermedia sketch canvas is intended to be low. In this view, designers can model the states and transitions, but cannot edit or view the response messages themselves. Further, the designer cannot model conditional logic or decision based state changes.

The Rapido canvas is meant to support quick, low fidelity sketches that require a minimum of effort and investment.

3.5 Editing API Message Data

Although the CRUD and hypermedia canvases do not support message level editing, designers are able to "zoom in" to a particular state or resource if they wish to sketch the message data.

Rapido's message editor is a simple text based editor with usability aides for JSON, HAL and Collection+JSON editing. This includes syntax highlighting and type-ahead hints for the supported formats (see Figure 10).

Designers are free to construct any text based response for the response or state node that they have chosen. While the editor may signal the validity of the data entered based on the media type's specification, it does not prevent the user from saving arbitrary data.

This allows designers to sketch the response data in broad strokes, capturing important ideas without being bound by the rules of syntax.

Figure 10 - Editing response data

In addition, designers of hypermedia APIs are able to use a short hand syntax to create transitions between states. When a string contains a token delimited by the strings **"$("** and **")"**, a state transition is automatically created. This facility allows designers to impact the low-fidelity canvas view of the system while sketching higher fidelity details.

A downside to providing a message editing facility is that it requires additional investment on the part of the API designer. This added design effort can reduce the disposability of the sketch.

3.6 Implementability

Rapido sketches can be exported into either the WADL or Blueprint service description formats. This makes it possible to use the implementer's preferred editor or tooling to create prototypes based on the sketch.

Figure 11 - Exporting a Rapido sketch

3.7 Benefits

The main strength of the Rapido tool is that it combines the holistic, low-fidelity, visual properties of the drawing method with the higher-fidelity literal properties of the writing method.

The visual nature of the sketching canvas also affords designers the ability to "play" with their designs, experimenting with different types of transitions and relationships.

Finally, the lack of support for dynamic responses, logical flows and message header detail may discourage designers from investing too much time in the sketch resulting in improved levels of disposability.

3.8 Limitations

While the low-fidelity of the Rapido tool helps designers create many disposable sketches, it prevents them from capturing important detailed characteristics of their APIs such as security controls, error handling and logical behavior.

Another challenge is that the focus on specialized support for CRUD conventions and hypermedia formats hinders broader experimentation. For example, it is difficult to use Rapido to design a new hypermedia media type or a new message format.

In addition, the current set of media types and service description formats is limited. This limits the usefulness of the tool to communities of users who use these particular formats. Hopefully in the future additional media types and formats will be supported.

Finally, many of the design decisions made for the Rapido tools are based on assumptions and hypothesis about user behaviour. In practice, the sketching needs of API designers may not be accurately reflected in the interaction design of the tool.

4. CONCLUSION

API designers who are interested in producing high quality interfaces should consider incorporating a sketching method into their design process.

While there are many different ways to sketch an API, the Rapido tool attempts to consolidate the benefits from the most popular methods to provide a sketching process with good disposability, experimentability, socializability and learnability.

By unbundling the sketching experience from the design, prototype and implementation experience Rapido attempts to provide a higher quality sketching experience for designers.

5. REFERENCES

[1] Joshua Bloch. 2006. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (OOPSLA '06). ACM, New York, NY, USA, 506-507. DOI=10.1145/1176617.1176622 http://doi.acm.org/10.1145/1176617.1176622

[2] Bill Verplank. 2009. Interaction Design Sketchbook.

[3] David G. Ullman, Stephen Wood, David Craig. The importance of drawing in the mechanical design process. Computers & Graphics 14, 2 (1990), 263--274.

[4] Drawing Gym, Teaching Engineers to Draw.: http://www.ucl.ac.uk/drawing-gym/. Accessed: March 9th, 2015

[5] Bill Buxton. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[6] Neil Cohn. 2012. Explaining 'I Can't Draw': Parallels between the Structure and Development of Language and Drawing.

[7] Marian Petre. 2013. UML in practice. In *Proceedings of the 2013 International Conference on Software Engineering* (ICSE '13). IEEE Press, Piscataway, NJ, USA, 722-731.

[8] Ole Lensmar. An Overview of REST Metadata Formats: http://apiux.com/2013/04/09/rest-metadata-formats/. Accessed: March 8th, 2015.

[9] Grady Booch, James Rumbaugh, and Ivar Jacobson. 1999. *The Unified Modeling Language User Guide*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[10] The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series) (13 October 1998) by Jos B. Warmer, Anneke G. Kleppe

[11] Ronnie Mitra, Rapido: http://www.rapidodesigner.com

[12] Apiary. http://apiary.io/. Accessed: March 9th, 2015

[13] I. Zuzak, I. Budiselic, and G. Delac. 2011. A FINITE-STATE MACHINE APPROACH FOR MODELING AND ANALYZING RESTFUL SYSTEMS.

[14] M. Amundsen, L. Richardson. 2012. RESTFul Web APIs. O'Reilly Media.