Community-based API Builder to manage APIs and their connections with Cloud-based Services

Romanos Tsouroplis[1], Michael Petychakis[1], Iosif Alvertis[1], Evmorfia Biliri[1], Fenareti Lampathaki[1], Dimitris Askounis[1]

[1] National Technical University of Athens, School of Electrical and Computer Engineering, Athens, Greece
{ rtsouroplis, mpetyx, alvertisjo, ebiliri, flamp, askous} @epu.ntua.gr

Abstract. Creating added value, innovative applications and services in the web is hindered by the prevailing different formats and models of information retrieved by the various Cloud-based Services (CBS). Additionally, CBS tend to change their Application Programming Interface (API) versions very often, not sufficiently helping the interested enterprises with their adoption as they need to be up-to-date and always functional. The API Builder is a community-based platform that aims to facilitate developers in adopting a Graph API that unifies the experience of multiple cloud-based services APIs and personal cloudlets, building and maintaining their software applications easily, despite any changes made in the CBS APIs.

Keywords: APIs, Cloud-based Services, Evolving APIs, Graph API, Community-based platform, Social Enterprise

1. Introduction

These days, more and more users tend to share their data through the World Wide Web. Social Networks have gathered large silos of information for each of their users and their interconnections. The Web APIs (Application Programming Interface) provide all the endpoints needed for a developer to request access into this plethora of information. All the major social networks like Facebook, Twitter, etc., have an API, providing most of their core services to third parties for several reasons [1].

Currently, on the site Programmable Web, which indexes, categorizes and makes mashups of APIs, 12,987 public APIs are listed and this number is continuously growing [2]. This, along with the fact that there is not a standardized, specific model representing each object in the world make the differentiations among the API objects a huge overhead for anyone that needs to integrate a product with various Cloud-based Services. In fact, there are so many services offering APIs for their nodes that a developer or even a team of developers within a company would have difficulties keeping up with the always evolving API landscape.

Companies, developers and end-users are directly affected by the great changes that CBS APIs go through continuously. In particular, Facebook API, one of the most used, is now on version 2.2 and all the client application using version 1.x have until

April 30, 2015 to upgrade to v2.0 or later, as they are going to be deprecated and not functional afterwards. Other APIs are developed keeping in mind that future versions will come, ensuring the end-users that what exists now will remain functional, like Dropbox claims to. Last but not least, many APIs become discontinued, like Desktop API by Skype. To sum up, APIs are born, evolve and die just like living organisms and there is need of great effort to keep up with them [3].

The effects that the Cloud-based Services API changes create on the developer community have been thoroughly researched [4, 5]. This is the main challenge we try to confront, so as to offer enterprises/developers the ease of mind that is needed for considering usage of one or multiple APIs when building their software products and services. The OPENi API Builder uses the power of community over a clean and simple user interface to produce an updated Generic API recommendation, based on valid REST API standards, matching the most used objects of a great number of Cloud-based Services APIs.

The structure of the present paper is as follows. Section 2 presents the methodology followed towards the implementation of the OPENi API Builder. In section 3, an overview of the API Builder is given, describing the added value features provided, broken up into 3 smaller sections, the community-based orientation, the documentation in all the standards currently available and lastly the case of CBS API changes handling. In section 4 we provide related work and compare it with the Builder. Section 5 concludes the paper with future work on this matter.

2. Methodology

Due to the high complexity of managing and orchestrating the changes in all of the CBS APIs, a mechanism that would handle these changes had to be implemented. But first, a research was conducted on the field of API creation. This was completed in 7 stages, based of course on the modeling already conducted for the Graph API [6, 7]. Validation of each step and peer-reviewing of the work done helped throughout the whole process to keep constantly up-to-date.

- **Step 1. Identifying the needs for handling the Graph API Evolution.** In this step, user stories were created to describe the expected functionalities for future updates on the Graph API Platform APIs, Objects, methods and CBS.
- **Step 2. Research of various similar solutions.** API creation tools can be found in many enterprise solutions across the Web, on studies and even as open source framework for specific code languages. All of these solutions were gathered and categorized based on the providing features.
- **Step 3. Connectivity with CBS.** The CBSs add extra value to a business solution as already described. Categorization on the CBS has already taken place for the Graph API Platform [6].
- **Step 4. Automation of the evolution process.** Research [3] has been conducted on the automation of the APIs evolution. There is no solution that does not involve examining the CBS backend code, so a semi-automatic approach was chosen based on a swift-responding to changes community.

- **Step 5. Building upon standards.** All of the available solutions depend on one or none of the thriving standards on API documentation. Support on the most significant of them was decided after carefully exploring them and their impact on the developer community.
- **Step 6. Identifying scalability concerns.** The Builder should be able to manage multiple simultaneous user actions and keep a huge amount of data for objects and interconnections with CBS, so an analysis of best practices towards this end was taken into consideration.
- **Step 7. Crafting UI mockups** to better express the power of the Builder. After assembling the requirements for making the API Builder, basic interface design principles like intuitiveness, familiarity, simplicity, availability and responsiveness were followed in order to ensure a great user experience.

3. Overview

The OPENi API Builder is a Web-based platform, implemented as an open source project, publicly available in Github [8].

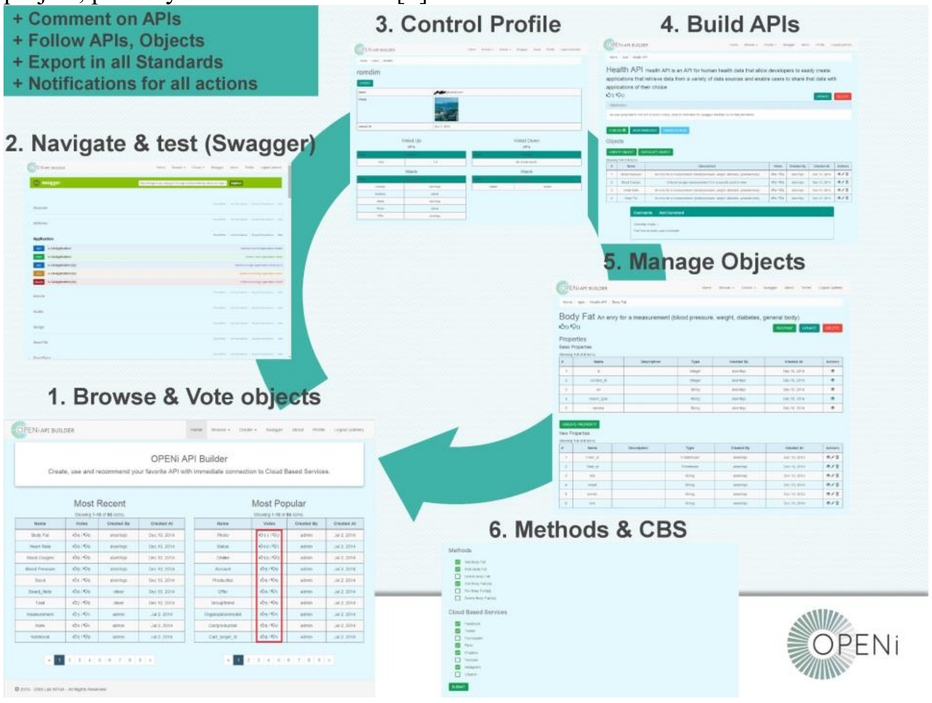

Figure 1 - API Builder Actions

Through its web interface, developers can create, update, delete their own APIs, duplicate existing ones, manage CBS API instances and create correspondences between them. In order to allow for more flexibility, APIs inside Builder consist of multiple components which can also be modified independently. More specifically, they have the following structure: APIs can have multiple Objects and Objects connect with Properties, Methods and CBS.

An authenticated developer can browse public APIs and Objects, vote them up/down, follow them or their creators as shown in Figure 1. API specification and testing is provided through a swagger interface embedded in the website. The provided level of automation in API handling helps developers focus on more meaningful tasks and deliver their work more quickly, increasing business profits.

The innovation of the API Builder can be found in 3 different features, the community backing the API and CBS management along with the social characteristics that are provided, the standards on which the Builder structures its documentation and the Cloud-based Service APIs evolution handling.

3.1 Community Orientation

The transformation of the World Wide Web these last years, from a static directory of connected but one-person authored files to the Social Web as we know it today, usually referenced as Web 2.0, showed the dynamics of the crowd in crafting truly valuable information. From the Linux Kernel [9] to the Microsoft .NET tools and frameworks, the open source solution gives the opportunity to build a large community around business products that make the end-users interact and bind with them. Many papers have been written for the advantages of transforming a traditional firm-based model into a community-based development process [10, 11].

The OPENi API Builder has been built to accommodate all the needs of an active community. In Table 1 we can see the actions that are allowed for an authenticated enterprise user. The Builder has a lot of social characteristics as well, allowing the users to interact and engage with each other as well as with their products. The influence indicator of an API for example can be calculated by taking into account the votes and the number of users following that particular API.

Builder	User Actions							Social Characteristics				
Parts	Create	Duplicate	View	Update	Delete	Select	Publish	Propose	Vote Up	Vote Down	Comment/Reply	Follow
API	Y		Y	Y	Y		Y	Y	Y	Y	Y	Y
Object	Y	Y	Y	Y	Y				Y	Y		Y
Property	Y		Y	Y	Y							
Method						Y						
CBS	Y		Y	Y		Y		Y	Y	Y		
Other Users			Y									Y

Table 1 - User Actions & Social Characteristics

In order to avoid duplicate declarations of Objects and promote reusability of objects inside the community, aiming at more sustainable APIs, during the creation of a new object the developer is presented with recommendations of existing ones that could be used instead, based on a combination of approximate string matching on the selected names and similarity computation of the types of objects.

3.2 Documentation based on standards

In order to provide a consistent way to document APIs, a number of REST metadata formats has been created. These standards offer a way to represent an API by specifying entry point(s), resource paths, methods to access these resources, parameters to be supplied with these methods, formats of inbound/outbound representations, status codes,error messages and documentary information.

Some of the most popular standards are: Swagger, which offers a large ecosystem of API tooling, has very large community of active users and great support in almost every modern programming language and deployment environment. API Blueprints, where an API description can be used in the apiary platform to create automated mock servers, validators etc. The Hydra specification [12], which is currently under heavy development, tries to enrich current web APIs with tools and techniques from the semantic web area. RAML is a well-structured and modern API modelling language. And finally WADL, an XML-based specification submitted to W3C, but not widely adopted. Table 2 show more information about these standards. Swagger is obviously the dominant choice for the moment, though all specs are promising.

In order to allow developers to interact with a wider range of APIs independently of their preferred standard, an API can be exposed through the Builder to any of the aforementioned specifications. Moreover, a format transformation component has been implemented, enabling transformation amongst these five specifications.

Details\Specs	API Blueprints	RAML	Hydra	WADL	Swagger
Format	Markdown	YAML	Hydra Core Voc. - JSON-LD	XML	JSON
Licence	MIT	ASL 2.0/TM	CC Atrribution 4.0	Sun Microsystems Copyright	ASL 2.0
Available	Github	Github	www.w3.org	www.w3.org	Github
Sponsored By	Apiary	Mulesoft	W3C Com Group	Sun Microsystems	Reverb
Version	Format 1A revision 7	1.0	Unofficial Draft 19 January 2015	31 August 2009	2.0
Initial Commit	April 2013	Sep 2013	N/A	Nov 2006	July 2011
Pricing Plans	Y	Y	N	N	N
StackOverflow Questions	75	37	35	156	732
Github Stars	1819	1058	N/A	N/A	2459
Google Search	1.16M	457k	86k	94.1k	28M

Table 2 – API Specification Details

3.3 CBS API changes handling

When a Cloud-based Service decides it is time to change its API, it may deliver additions, deletions and/or alterations on existing Objects and/or the url schema itself. This is a huge overhead for enterprises as specified in the introductory section. It can lead to great increase of the development cost to cover new CBS documentation and code refactoring. The OPENi API Builder provides, to the best of our knowledge, the easiest way to adapt all these changes through the swift reflexes of its community, hence saving time and expenses from enterprises.

All the Objects created at the Builder can have their properties mapped to the corresponding properties of a similar Object of some CBS. These mappings are essentially arrays of labels, where each label represents a property from this particular CBS API. Following version changes is as easy as drag and dropping new labels, removing the deprecated ones and renaming those that need to be altered. Extra attention needs to be given though to the required properties of an Object and the methods that can be applied to each Object. Through a clean user interface, the process of making a match with an API is easier than using wrapper code in some programming language. Even the url that is needed to make the calls can be provided and altered in plain text.

When a new matching version is ready, developers can propose this API for implementation in the official OPENi Platform, and create the documentation in all available specifications. Through the Builder social characteristics, the community advances the most used and up-to-date APIs, promoting them for enterprise solutions.

4. Related Work

Several alternative business and developer-oriented solutions exist for creating and managing APIs, based on various or no standards. StrongLoop Arc provides a nice user interface for implementing APIs that are then rendered via a Swagger-based API explorer. Alas, there is no community built upon this solution. Apiary uses API Blueprints code to provide the documentation along with additional features. The downside is that the API management is accomplished via the code. The Apiary solution has a community but no social characteristics. API Designer by Mulesoft has no visual interface as well. The developer writes RAML code to create an API. Appcelerator, a company which targets mobile apps, gives an interface for rapid assembly and hosting of mobile-optimized APIs with some prebuilt connectors. Apigility, an open source tool created by Zend Framework, running on own hosting environment, has visual interface for creating APIs with the ability to produce documentation in Swagger. Apigee Edge API Platform innovates in providing good analytic services for the APIs while providing most of the available CBS connectors. And last but not least, Kimono, by kimonolabs is a very interesting tool which allows users to scrape a website, by simply selecting data on the site. The tool then gathers similar information from the site into an endpoint for a new API. It does not have the same flexibility in managing your API but instead tries to automate the process in existing websites. No API documentation standard is yet provided. All of the feature variations can be found in Table 3 below.

Almost all of these solutions lack the social characteristics that the Builder provides. The most important differences though are the support of multiple available standards for documentation and the mapping between Cloud-based Services and the created APIs that keep track of all the changes in versions.

Features\Solutions		StrongLoop Arc	Apiary	API Designer	Appcelerator	Apigility	Apigee	kimono	API Builder
	user interface	Y			Y	Y	Y	Y	Y
	open source					Y			Y
	community		team-oriented	Y	team-oriented			Y	Y
	social network		Y	Y				Y	Y
Specification	Swagger	Y				Y	tweaked ui		Y
	Hydra								Y
	RAML			Y					Y
	WADL								Y
	API Blueprints		Y						Y
	Datastore Connectors	Y					Y		Y
	Connectivity with CBS				static		Y		dynamic
	Matching with CBS								Y

Table 3 - Related Work

In this context, the User Interface is the visual representation of the APIs management board through buttons, textboxes and labels. When no UI is provided, coding is used for APIs administration. The only open source alternative to the API Builder is Apigility. All the other Platforms are products developed by companies and have various pricing plans. Most of them have community features, though in Apiary and Appcelerator it has the form of a team-based product with cooperation features varying according to the pricing plans. Apiary, API Designer and kimono have social features like commenting, following and liking other projects. For Apiary this can be seen only inside the team collaborating on a project. Extra attention has been given to the OPENi API Builder on this matter as already described before in section 4.1.

For the documentation standards supported, StrongLoop Arc and Apigility follow Swagger, while Apiary is based on Api Blueprints and API Designer on RAML. Apigee is among the first ones supporting Swagger. To the best of our knowledge, only the API Builder supports all forms of standards for the documentation. Datastore connectors can be all sorts of SQL and NoSQL databases, like MySQL, MongoDB, SQL Server, etc. StrongLoop Arc and Apigee support integration and retrieval of these databases' models for creating APIs as well as the API Builder.

Connectivity with CBS is predefined for Appcelerator for a range of enterprise and public data sources like SAP, Oracle, Facebook, Twitter, etc. In the API Builder the Cloud-based Services can be altered dynamically through the community. Last but not least, the feature of matching Objects of APIs to other Objects from CBS can only be met at the API Builder as described before in section 4.3.

5. Conclusion

OPENi API Builder delivers a unique proposition for enterprises that want to go the extra mile in getting ahead of their competition. Utilizing the power of the community for seamless integration with multiple Cloud-based Services, the API Builder has a great value as a platform for enterprises, start-ups and freelancers. There is no need to keep track of the latest changes in CBS APIs, reassuring continuous uptime in own services. Easy to use and clear, Builder delivers documentation for its APIs in the most popular standards, providing compatibility with most business needs. OPENi API Builder, as a work in progress has to be further expanded and tested to fully discover its potentials and extend it towards the community's needs. The code is open source, GNU licenced and can be found at Github repository.

As future work we intend to disseminate the power of the Builder and increase its usage among developers/companies. We are also exploring the automated management of Cloud-based Services through different versioning, by parsing a documentation in one of the standards to automatically create the whole API in the Builder. This has already been implemented for Swagger documentation for the needs of interconnection between the Graph API Platform and the API Builder. Since the major documentation standards can be transformed from one to another as described in section 4.2, what remains is to determine the versioning and deprecation policy required to guarantee the most comforting solution for developers, start-ups and enterprises when using the API Builder.

Acknowledgments. This work has been created in the context of the EU-funded project OPENi (Open-Source, Web-Based, Framework for Integrating Applications with Social Media Services and Personal Cloudlets), Contract No: FP7-ICT-317883.

References

1. A Survey of the API Economy. Israel Gat, Giancarlo Succi. Agile Product & Project Management. Executive Update Vol. 14, No. 6
2. Programmable Web API Directory. http://www.programmableweb.com/apis. [Accessed: 28-02-2015]
3. Espinha, Tiago, Andy Zaidman, and Hans-Gerhard Gross. "Web api growing pains: Stories from client developers and their code." *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 2014.
4. Wang, Shaohua, Iman Keivanloo, and Ying Zou. "How Do Developers React to RESTful API Evolution?." *Service-Oriented Computing*. Springer Berlin Heidelberg, 2014. 245-259.
5. Robbes, Romain, Mircea Lungu, and David Röthlisberger. "How do developers react to api deprecation?: the case of a smalltalk ecosystem." *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012.
6. Alvertis I., Petychakis M., Lampathaki F., Askounis D., Kastrinogiannis T. " A Community-based, Graph API Framework to Integrate and Orchestrate Cloud-Based Services." AICCSA. 2014
7. Petychakis, M., Alvertis, I., Biliri, E., Tsouroplis, R., Lampathaki, F., Askounis D. "Enterprise Collaboration Framework for Managing, Advancing and Unifying the Functionality of Multiple Cloud-Based Services with the Help of a Graph API." *Collaborative Systems for Smart Networked Environments* (2014): 153-160.
8. OPENi API Builder source code at Github, https://github.com/OPENi-ict/api-builder
9. Lee, Gwendolyn K., and Robert E. Cole. "From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development." *Organization science* 14.6 (2003): 633-649.
10. Shah, Sonali K. "Community-based innovation and product development: findings from open source software and consumer sporting goods." (2003).
11. Krishnamurthy, Sandeep. "An analysis of open source business models." (2005).
12. Lanthaler, Markus, and Christian Gütl. "Hydra: A Vocabulary for Hypermedia-Driven Web APIs." LDOW. 2013. APA