A Data-Driven Approach to Improve the Process of Data-Intensive API Creation and Evolution

Alberto Abelló, Claudia Ayala, Carles Farré, Cristina Gómez, Marc Oriol, and Oscar Romero

Universitat Politècnica de Catalunya, BarcelonaTech

{aabello,cayala,farre,cristina,moriol,oromero}@essi.upc.edu

Abstract. The market of data-intensive Application Programming Interfaces (APIs) has recently experienced an exponential growth, but the creation and evolution of such APIs is still done ad-hoc, with little automated support and reported deficiencies. These drawbacks hinder the productivity of developers of those APIs and the services built on top of them. In this exploratory paper, we promote a data-driven approach to improve the automatization of data-intensive API creation and evolution. In a release cycle, data coming from API usage and developers will be gathered to compute several indicators whose analysis will guide the planning of the next release. This data will also help to generate complete documentation facilitating APIs adoption by third parties.

Keywords: API, Data flows, Data management

1 Introduction

Nowadays, the use of Application Programming Interfaces (APIs) is crucial in technology-driven businesses as they permit to expose and use (through standardised protocols) diverse functionalities and data. APIs represent a powerful opportunity for companies and developers not only to leverage business value and simplify the software development lifecycle but also to promote innovation [19].

From the wide API marketplace, we are interested on those APIs that extract huge amounts of data from their organization repositories and share it with its customers. We refer to them as *data-intensive APIs*, drawing an analogy with the same concept in other fields (e.g., data-intensive applications [8]). We may find many success stories, like Smart Cities, where data-intensive APIs are a means to share data with end-users.

However, the rapid proliferation of data-intensive APIs has not been accompanied by sufficient support in their engineering process and several challenges still remain. In particular, there are neither creation nor evolution approaches considering their specificities. The generation and evolution of the data flows to extract the company's data from its internal data sources to be exposed through APIs is a complex and cumbersome task that may render unfeasible when facing Big Data scenarios (such as those of Social Networks or Smart Cities). Moreover, existing research focused on how to create and

X. Franch, J. Ralyté, R. Matulevičius, C. Salinesi, and R. Wieringa (Eds.):
CAiSE 2017 Forum and Doctoral Consortium Papers, pp. 1-8, 2017.

evolve APIs does not consider: 1) monitoring techniques to extract extensive API usage information; 2) studying the software repositories used by developers in applications that use the APIs; 3) analyzing explicit feedback given by end users and software developers; 4) using techniques in the API creation process that favour their later evolution; 5) systematically generating and evaluating alternative API evolution plans.

To overcome these limitations, we explore in this paper the appropriateness of using a data-driven approach to support the creation and evolution of data-intensive APIs. This approach is based on the exploitation of the data sources mentioned above (i.e., software monitors, user feedback and exposed data repositories) through highly automated processes. To this end, within a release cycle, data coming from both API usage and developers (both monitored and provided explicitly) will be gathered and analysed in order to compute several indicators whose analysis will guide the planning of the next release. This data will also be used to improve API documentation, including specific non-functional requirements and usage pattern examples, that will facilitate the adoption of those APIs by third parties.

The rest of the paper is structured as follows. Section 2 reviews the state of the art in API creation and evolution and presents the challenges in the area. Section 3 describes the generic data-driven approach that we are envisaging. Section 4 discusses an example of use of this approach in the Smart Cities context and Section 5 summarises with the relevant conclusions.

2 Challenges in Data-Intensive API Creation and Evolution

In the next subsections, we summarise the state of the art in API creation and evolution, and the challenges when considering the specific case of data-intensive APIs.

2.1 API Creation

Recent think tanks have advocated for automating data sharing and management in order to democratise the access to information [9]. The Open Data paradigm builds upon this idea and, aligned with it, a new market of external data is already available, which represents a potential wealth of data to be used by data-intensive APIs. However, the current state of the art on API creation ignores this type of data sources [2] and, in general, how to deal with huge amounts of data in the API creation process. Considering external data means extracting, integrating and consolidating them in order to be efficiently exposed. These tasks are part of data-intensive flows (DIF) management, massively studied in the Business Intelligence (BI) arena. Nevertheless, DIFs are still a key challenge for next generation BI systems, which advocate for empowering end users in the management of data [11].

Currently, some of the works related to API creation deal with the modelling of the static structure of the interfaces and/or their behaviour [4, 2, 14, 12, 10, 24]. These aim to automate API creation but derived APIs and their implementation do not take into account their posterior evolution.

2.2 API Evolution

Most of the recent state of the art on API evolution consists of empirical studies that address their impact on the consumer side (i.e., on the software developers and applications that use the evolving APIs) [23, 5, 21, 25].

The core of the problem in API evolution, however, is still to determine what, how, and when to evolve. Therefore, it seems necessary to 1) collect runtime data from API requests, and 2) decide, based on the analysis of such data, which changes are required and how they are prioritised, scheduled, and enacted. Optimization-based approaches proposed in the related field of software release planning [1] may be appropriate also in the context of API evolution. To this end, the definition of key-performance indicators may be of interest to compare alternative strategies [15].

The feedback of the community of software developers who use the APIs should also be taken into account when planning their evolution. There are some examples of methods and tools that allow users participate actively in this, e.g., My Experience [6], play.tools [22], Requirements Bazaar [16] or the commercial UserVoice[1]. In any case, the users in these examples are end-users of applications or devices. To the best of our knowledge, there is no initiative that takes into account the developers of applications and services that have a high-stake interest in the data-intensive APIs that they use and are able, at the same time, to provide high-quality technical feedback (e.g., on bugs or missing features) that end users cannot.

2.3 API Documentation

An important aspect related to both API creation and evolution is the generation and evolution of a comprehensive and accurate documentation. Existing works on API documentation have suggested that, in order to successfully use APIs, it is required that their documentation includes explanations and examples of their use [17]. Poor and incomplete documentation is one of the main obstacles faced by developers to understand and use new APIs [17]

Usually, API documentation is generated manually and there is a lack of support for evolving such documentation as APIs evolve over time. Therefore, API documentation becomes obsolete quickly [17]. Several tools, such as JavaDoc[2], Swagger[3], Blueprint[4] and SpyREST [20] have been developed to automatically generate and maintain up-to-date API documentation, and some contributions have been made to improve it [18]. However, other studies [13] reveal that there is still much room for improvement. [20] proposes to generate and maintain API documentation by first monitoring and then analysing the requests to the API. However, the limitation is clear: the process of synthesizing the requests needed to infer the documentation is manual, with the risk of not covering all possible cases. Moreover, it also requires knowledge of the API that obviously has not been acquired through its documentation.

---

1 https://www.uservoice.com

2 http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html

3 http://swagger.io

4 https://apiblueprint.org

2.4 Challenges in Data-Intensive API Creation and Evolution

In this section, we detail the challenges to be faced by the data-driven approach for creating and evolving data-intensive APIs.

Ch1. Defining data-intensive API creation for evolving. Most of the existing works related to API creation deal with the modelling of their static structure and behavioural part without considering their later evolution. Moreover, these works do not take into account the specificities of data-intensive APIs. The complexity of data-intensive API evolution requires new techniques and methods that consider future evolution due to changes in both data sources and requirements.

Ch2. Improving the automatic generation and maintenance of data-intensive API documentation. A complete, accurate and updated documentation is a key factor for the wide use of data-intensive APIs. Currently, the quality, maintenance and the automatic generation of API documentation are in question. So, new approaches are needed to improve the automatic generation and maintenance of data-intensive API documentation, which includes the internal transformations of data.

Ch3. Exploiting available evidence to generate data-intensive API evolution proposals. The observations provided by data-intensive API users (developers of software applications that consume those APIs and end-users of applications using such APIs) as well as the analysis of data-intensive API usage and the software repositories of applications that consume those APIs, are valuable information that needs to be examined in more detail to generate data-intensive API evolution proposals.

Ch4. Defining indicators to evaluate data-intensive API evolution proposals. When several evolution proposals emerge, it is difficult to choose among them the ones to be implemented. The challenge is to define a set of strategic indicators that helps to evaluate and prioritize the implementation of those proposals.

Ch5. Automating the data-intensive API evolution process. The state of the art in API evolution shows that it is still quite manual. To deal with API evolution challenges we will focus on exploiting information from diverse sources such as monitors, software repositories, and explicit feedback.

3 A Data-Driven Approach to the Creation and Evolution of Data-Intensive APIs

As we have seen in the previous section, the process of creation and evolution of data-intensive APIs has still much room for improvement. Increasing their automated support would improve the efficiency and effectiveness of the API engineering process. Thus, we propose a data-driven approach aiming to improve the process of creation and evolution of data-intensive APIs. We label our proposal as “data-driven”, because its formulation relies on gathering and analysing data coming from all the sources mentioned in Section 2 (i.e., organization databases, software repositories, API usage, etc.). We call ***Smart APIs*** those data-intensive APIs that have been generated and evolved using the data-driven approach explored in this paper, and define three separate processes (shown in Figure 1) in order to automate their creation and evolution.

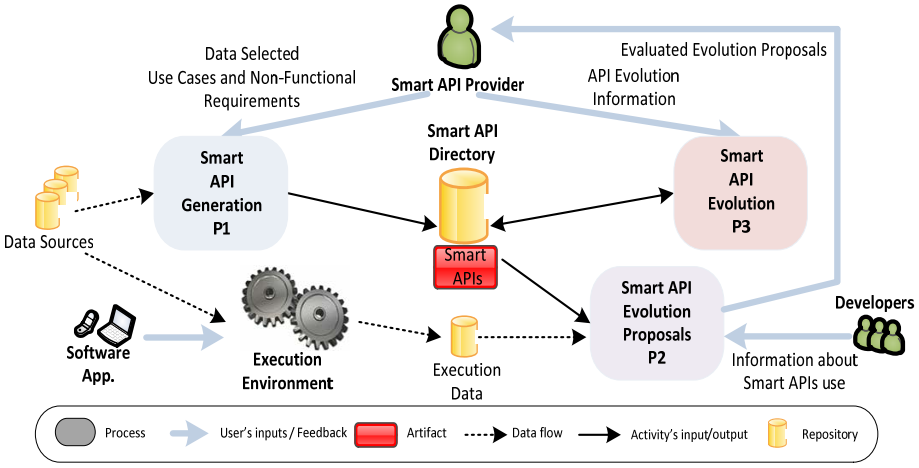

Fig. 1. General overview of a data-driven approach to Smart API creation and evolution

P1. Smart API Generation. This process aims at gathering the information, metadata and behaviour required to create a Smart API to be published by the API provider. With this information, a Smart API, its documentation with related metadata (including non-functional requirements, NFRs for short) and its implementation will be generated and stored in a directory. Additionally, this process will automatically generate and manage the DIFs mentioned in Section 2.1 and an ad-hoc API DB required to efficiently consume relevant external data. This process will address challenges Ch1 and Ch2 described in the previous section.

P2. Smart API Evolution Proposal Generation. As Smart APIs are invoked in an execution environment (that can be obtained, for instance, from transformations applied within the Model Driven Development (MDD) paradigm [7]), they will store data coming from their execution. Information that will be generated at runtime (e.g., response time of the Smart APIs), obtained from the developers who use the Smart APIs (e.g., detection of bugs), accessed through the software repositories of applications that consume the Smart APIs (e.g., sequence of APIs invoked) and the information provided by the Smart API users (developers and end-users) will be processed to recommend candidate evolution proposals for the Smart APIs. Those proposals, which will be evaluated using a set of strategic indicators, will be rendered to the Smart API provider who will be able to make informed decisions. This process will address challenges Ch3 and Ch4.

P3. Smart API Evolution. The last process will automate the Smart API evolution from the proposals obtained in P2. The API provider will select the Smart APIs that wants to evolve and provide the additional information needed to implement it. This process will finally evolve the Smart APIs, and the entries in the directory of the corresponding APIs will be modified regenerating the data flows and API DB required to the new version of the APIs. This process will address challenge Ch5.

4 Example of Application: Smart APIs for Smart Cities

Smart Cities is one of the domains where the use of data-intensive APIs is becoming commonplace. Initiatives such as FIWARE[5], CitySDK[6] and iCity[7], all of them with public funding from European programs, propose API-centric platforms and standards. The goal of these initiatives is to enable citizens and organizations to access Smart City data and services through applications that consume such APIs. The confluence of huge amounts of open data, e-government and real-time information from millions of sensors and the citizens themselves, accessible via APIs, should contribute to a more efficient and sustainable operation of Smart Cities [3].

Thus, Smart Cities show most of the challenges discussed in Section 2. For example, Boyd refers to improving documentation as one of the challenges to improve the usability and reliability of APIs [3], especially important in the Smart City context. To illustrate how these challenges are currently being faced in these APIs, we describe below some of the limitations and problems that we experienced when developing some mobile apps exploiting the APIs provided by a city Smart City infrastructure[8].

- **Inconsistencies**. Some of the APIs did not provide exactly the functionality that they promised in the documentation. When we contacted with the Smart City department and discussed the problem, they provided two main reasons for this: 1) simply, they made some mistake due to the lack of tool support in the API creation; 2) at some moment, some evolution in the data sources behind the API was not propagated to the API itself (because the data source was not managed by the Smart City department). Converting these APIs to Smart APIs would have avoided these problems due to the generation of the APIs from the artefacts specified by the Smart City department (acting as Smart API provider).
- **Documentation**. In general, as reported in Section 2, documentation was very rudimentary, hard to follow (examples were scarce), and sometimes incomplete or even inaccurate. For instance, some measurements of IoT sensors did not specify the units of the provided data. Also, we discovered that some of the parameters in the methods provided by the APIs were not yet implemented at that time, but only planned for next releases, with no clear indication in the documentation. Service level agreements were not specified, not allowing then to state reasonable NFRs, as for instance response time, from the customer side since it was going to be not possible to check them.
- **Feedback**. During the usage of APIs along some months, we identified and reported some bugs in the system. For instance, some IoT sensors didn't return measurements reporting a time out. Such bug reports were done manually by e-mail, which was a first hurdle to us. It was the case that we experienced some delays on the responses, and we inquired the process followed: our reports were dispatched to the team of

[5] www.fiware.org
[6] www.citysdk.eu
[7] www.icityproject.eu
[8] We cannot disclose the name of the city due to confidentiality reasons.

developers, then team worked on the solution, and when fixed, they reported back to our interlocutor, who informed us. All these steps made the interaction slow and cumbersome, leading to a slow non-interactive process resolution. Furthermore, the problem scaled up when the issue was related to third-party APIs (i.e. APIs for which the Smart City systems acted as customers), such as when the Smart City used third-party sensors: any reported issue took weeks before receiving any feedback. Sophisticated feedback channels but especially, the replacement of explicit feedback by API usage monitoring, should lead to dramatic improvements in the quality of service delivered to customers and the time to fix any issue.

- **Evolution**. With respect to API evolution, the platform provided a forum with several discussion channels, to potentially get feedback. However, the forum did not attract many users, making it unsuitable for capturing user feedback enough to help evolve the API. Therefore, at the end, evolution decisions were not really driven by real opinions. We envision that with the data-driven approach proposed in this paper, the developers will get the feedback required to help them identify the right direction to evolve the API, not only from explicit feedback from the opinions of the users but also from implicit indicators obtained by monitoring how the APIs are used.

5 Conclusions

In this exploratory paper, we have proposed a novel data-driven approach to improve the creation and evolution process for data-intensive APIs with the goal of overcoming the limitations reported in the state of the art in this area and responding thus to some resulting challenges. We have presented a general conception of the approach. Possible instantiations may emphasise the link between the APIs and the data sources through elaborated traceability links, or use goal models as a way to elicit and analyse the optimal features to include in the APIs.

Acknowledgements

This work is funded by the Spanish project TIN2016-79269-R, AEI/FEDER, UE.

References

1. D. Ameller, C. Farré, X. Franch, G. Rufian. *A Survey on Software Release Planning Models*. In PROFES 2016
2. C. Alexopoulos, E. Loukis, Y. Charalabidis. *A Platform for Closing the Open Data Feedback Loop Based on Web 2.0 functionality*. In eJournal on eDemocracy and Open Government 6(1), 2014
3. M. Boyd. *How Smart Cities Are Promoting API Usage*, ProgrammableWeb. 04-05-2015. Available: http://www.programmableweb.com/news/how-smart-cities-are-promoting-api-usage/analysis/2015/05/04. [Accessed: 21-Nov-2016]
4. H. Ed-douibi, J. L. Cánovas, A. Gómez, M. Tisi, J. Cabot. *EMF-REST: Generation of RESTful APIs from Models.* In SAC 2016

5. T. Espinha, A. Zaidman, H.-G. Gross. *Web API growing pains: Loosely coupled yet strongly tied*. In Journal of Systems and Software, vol. 100, 2015
6. J. Froehlich, M.Y. Chen, S. Consolvo, B. Harrison, and J.A. Landay. *MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones*. In MobiSys 2007
7. M. Völter, T. Stahl: *Model-Driven Software Development: Technology, Engineering, Management.* Wiley and Sons, New York, USA, 2006.
8. D. Habich, W. Lehner, S. Richly, U. Aßmann: *Using Cloud Technologies to Optimize Data-Intensive Service Applications*. In IEEE CLOUD 2010
9. *Unleashing the Potential of Big Data*. White paper based on 2013 World Summit on Big Data and Organization Design. Available: http://www.e-pages.dk/aarhusuniversitet/775
10. A. Ivanchikj, C. Pautasso, S. Schreier. *Visual modeling of RESTful conversations with RESTalk.* In SSM 2016.
11. P. Jovanovic, O. Romero, A. Abelló: *A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey*. Trans. Large-Scale Data- and Knowledge-Centered Systems, 28 (5), 2017
12. L. Li, W. Chou. *Designing Large Scale REST APIs Based on REST Chart*. In ICWS 2015
13. M. A. Saied, H. Sahraoui, B. Dufour. *An Observational Study on API Usage Constraints and Their Documentation*. In SANER 2015
14. I. Porres, I. Rauf. *Modeling Behavioral RESTful Web Service Interfaces in UML*. In SAC 2011
15. A. M. Pitangueira, P. Tonella, A. Susi, R. S. Pitangueira Maciel, M. de Oliveira Barros: *Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization.* In REFSQ 2016
16. D. Renzel, M. Behrendt, R. Klamma, and M. Jarke. *An open requirements bazaar for social requirements engineering in the long tail*. In RE 2013
17. M. P. Robillard and R. Deline. *A field study of API learning obstacles*. In Empirical Software Engineering 16(6), 2011
18. S. Radevski, H. Hata, K. Matsumoto. *Towards Building API Usage Example Metrics*. In SANER 2016
19. Sandoval Kristopher Sandoval. *Leverage Public APIs to Make Your System Better*. 16-11-2016. Available: http://nordicapis.com/leverage-public-apis-to-make-your-system-better
20. S. M. Sohan, C. Anslow, and F. Maurer. SpyREST: Automated RESTful API Documentation using an HTTP Proxy Server. In ASE 2015
21. S. M. Sohan, C. Anslow, F. Maurer. *A Case Study of Web API Evolution*. In ICWS 2015
22. C. Schueller and W. Woerndl. *Automated User Feedback Generation in the Software Development of Mobile Applications*. In Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg, Nr. 42, 2008
23. S. Wang, I. Keivanloo, Y. Zou. *How Do Developers React to RESTful API Evolution?* In ICSOC 2014
24. C. Zolotas, T. Diamantopoulos, K. C. Chatzidimitriou, A. L. Symeonidis. *From requirements to source code: a Model-Driven Engineering approach for RESTful web services*. In ASE 2016
25. A. V. Zarras, P. Vassiliadis, I. Dinos. *Keep Calm and Wait for the Spike! Insights on the Evolution of Amazon Services*. In CAISE 2016