The Journal of Systems & Software 198 (2023) 111609

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Web API evolution patterns: A usage-driven approach☆

Rediana Koçi *, Xavier Franch, Petar Jovanovic, Alberto Abelló

Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Catalunya, Spain

ARTICLE INFO

Article history:
Received 26 May 2022
Received in revised form 23 September 2022
Accepted 2 January 2023
Available online 5 January 2023

Keywords:
Web API evolution
Web API logs
Process mining
Usage patterns

ABSTRACT

As the use of Application Programming Interfaces (APIs) is increasingly growing, their evolution becomes more challenging in terms of the service provided according to consumers' needs. In this paper, we address the role of consumers' needs in WAPIs evolution and introduce a process mining pattern-based method to support providers in WAPIs evolution by analyzing and understanding consumers' behavior, imprinted in WAPI usage logs. We take the position that WAPIs' evolution should be mainly usage-based, i.e., the way consumers use them should be one of the main drivers of their changes. We start by characterizing the structural relationships between endpoints, and next, we summarize these relationships into a set of behavioral patterns (i.e., usage patterns whose occurrences indicate specific consumers' behavior like repetitive or consecutive calls), that can potentially imply the need for changes (e.g., creating new parameters for endpoints, merging endpoints). We analyze the logs and extract several metrics for the endpoints and their relationships, to then detect the patterns. We apply our method in two real-world WAPIs from different domains, education, and health, respectively the WAPI of Barcelona School of Informatics at the Polytechnic University of Catalonia (Facultat d'Informàtica de Barcelona, FIB, UPC), and District Health Information Software 2 (DHIS2) WAPI. The feedback from consumers and providers of these WAPIs proved the effectiveness of the detected patterns and confirmed the promising potential of our approach.

1. Introduction

Application programming interfaces (APIs) provide an abstraction layer built upon sets of low-level methods and functions. API providers build these methods and functions and offer their interfaces so that API consumers (i.e., software developers) can easily use them in their applications. While applications locally access the 'traditional' APIs (i.e., statically linked APIs), web APIs (WAPIs) are exposed over the Internet, hence their endpoints (URLs to WAPI resources) are remotely accessed.

Being provided, and thus accessed remotely, imposes challenges for both parts, notably during WAPI evolution. Consumers are obliged to update their applications under WAPI evolution pace (Wang et al., 2014; Li et al., 2013; Espinha et al., 2014, 2015). On the other hand, providers have their own burden: striking a balance between not imposing unexpected, frequent changes and providing an up-to-date, maintainable, bug-free WAPI, that fulfill consumers' needs and business (organizations that are exposing the data) requirements. Knowing the considerable impact WAPIs have on consumers, providers would benefit from their feedback to better understand their needs (Murphy et al., 2018). Consequently they can implement changes directly targeting consumers' concerns.

If consumers encounter any bugs, or they need a new feature implemented in the WAPI they are using, they commonly report the bug or request the new feature to WAPI providers by means of issue tracker systems, communities of practice (e.g., https://community.dhis2.org) or other communication channels between providers and consumers (explicit feedback given to providers). However, due to the lack of these communication channels or because of consumers' neglect to report, several WAPI bugs remain unreported and several desired features or improvements unlisted and undiscovered (Lamothe and Shang, 2020).

Actually, as WAPIs are exposed over the network, providers can observe the way consumers use them, and can indirectly track consumers' needs. Furthermore, they can detect flaws related to the usability of the WAPIs, of which consumers may not be easily aware. For instance, in large companies with separated WAPI development teams for each domain (i.e., for marketing, sales, product, and customer service), it may happen that, for the same functionalities, several WAPI endpoints are designed. The way consumers access them, may reveal their redundant, duplicated, or combinable functionalities.

We consider consumers' implicit feedback as the fuel of this process. Assuming that WAPI evolution is and should be driven by

☆ Editor: Neil Ernst.

* Corresponding author.
E-mail addresses: koci@essi.upc.edu (R. Koçi), franch@essi.upc.edu (X. Franch), petar@essi.upc.edu (P. Jovanovic), aabello@essi.upc.edu (A. Abelló).

https://doi.org/10.1016/j.jss.2023.111609

the way WAPI is consumed, we propose a data-driven approach to anticipate changes based on consumers' behavior. Therefore, consumers' behavior is recorded in these logs, and their analysis can find room for potential improvements, or obliquely reveal consumers' needs for new features hidden under several workarounds (solutions found by WAPI consumers that allow them to get data, functionality, or features they need, but that are not yet implemented by providers). There are several WAPI monitoring tools available, but they are mostly oriented toward providing reporting dashboards or automatic alerting in case of WAPI failure (Doerrfeld, 2018). Consequently, WAPI providers have all this potentially insightful, large volume of data that is being generated, but not proactively used for evolution.

Here is where process mining promises to be useful, providing a set of well-established analysis techniques to extract process-related insights from event logs (Van Der Aalst, 2016). By making use of the WAPI usage data available on the providers' side, and the applicability of process mining in web services context (Van Der Aalst, 2012, 2013), we can analyze the way consumers use the WAPI, and identify behavioral patterns (usage patterns that indicate specific behaviors) that can imply the need for change. Additionally, having all the usage scenarios from consumers for the endpoints that will be modified, providers can provide better migration scripts to help them migrate to new WAPI versions.

This paper is in the same line as our previous works, where we target WAPI evolution and leverage the availability of WAPIs usage logs. Previously, we analyzed WAPI development usage logs (logs generated when consumers are developing and testing their applications) (Koçi et al., 2020) and prescribed a set of usability-related changes. Then, we focused our research on production logs analysis (generated after consumers' applications' release) and presented our preliminary results through a tool that aims to support providers in planning the changes (Koçi et al., 2021b).

Building upon these previous results, in this paper we extend the goal of Koçi et al. (2021b) and elaborately introduce a method to help providers in the evolution of their WAPIs. We keep the position that WAPIs evolution should be mainly usage-based, i.e., the way consumers use WAPIs should be one of the main drivers of WAPI changes (Abelló et al., 2017). To this aim, we first characterize the relationships (e.g., consecutive calls) between two endpoints (URLs to access API resources). Specific relationships may hint several behavioral aspects, indicating room for potential improvements (e.g., merging endpoints), the presence of workarounds, or other reasons for performing changes to the WAPI. We summarize these relationships into a set of behavioral patterns whose occurrences suggest the need for specific WAPI changes. Then, we analyze the usage logs to elicit the actual use of the WAPI and detect the occurrences of the pre-defined patterns. Employing process mining techniques, we detect from the logs snippets of the entire process model with WAPI-related semantics, in the form of several metrics for the endpoints and their relationships. We use these metrics to later detect frequent patterns.

The main contributions of the paper are as follows:

- We define and present a set of WAPI behavioral patterns, whose occurrences in the logs indicate the need for WAPI changes.
- We introduce several useful metrics following process mining techniques in order to detect the occurrence of patterns in the logs.
- We demonstrate the applicability of our approach by exemplifying it on real-world usage logs from two different WAPIs: (i) District Health Information Software 2, and (ii) Barcelona School of Informatics at the Polytechnic University of Catalonia (Facultat d'Informàtica de Barcelona, FIB, UPC).
- We show the significance of the detected patterns and the feasibility of the proposed changes, by interviewing the WAPI consumers and providers to directly assess the results.

2. Background

In this section, we introduce the primary concepts used throughout this paper. We start with describing the main input of our approach, namely the WAPI usage logs, over which we apply process mining techniques. We explain some of their general definitions and then interpret them in the WAPI case. After that, we introduce graphlets, a concept that helps us in specifying the structural relationships between endpoints, which we later use to define the set of patterns.

2.1. WAPI usage logs

Providers typically log their WAPIs' usage by recording all requests done against the WAPIs. Every time a consumer issues a request to a WAPI, a log entry is generated and stored in the usage log file. The information that is logged for each request and its format may vary due to different logging system setting parameters and also providers' decisions about logs design. For example, using the Apache Custom Log Format,[1] (a flexible and customizable format), when an application makes a request like https://api.fib.upc.edu/v2/sales?client_id=ID to an WAPI endpoint, the following information could be logged by providers (based on the logging system configuration): the IP address of the application's user, the timestamp when the consumer made the request, the request method (`GET`), endpoint ($v2/sales?client\_id = ID$) the protocol (`HTTP/1.1`), the time needed to respond to the request, the status code (e.g., 200 if the request was successful), the size of the object returned, the address of the page that initiated the request, information about the operating system or browser used, and other fields comprising different aspects of consumers' applications and their users (e.g., `10.28.120.115 [24/Sep/2018:16:00:03 +0200] GET v2/sales?client_id=ID) HTTP/1.1 116 200 9436 "https ://..". "Mozilla/5.0..."`).

Based on consumers' applications lifecycle, we distinguish two different types of WAPI usage logs: (i) development logs generated at design time, and (ii) production logs generated at runtime. We point out this distinction as each of these logs owns specific characteristics and elements of interest, and can be used to analyze consumers' behavior from different aspects.

Development logs are generated while developers build and test their applications. Thus, these logs show their attempts in using the WAPI, the endpoints they struggle more with, or specific mistakes they make while using and learning the WAPI (Koçi et al., 2020; Macvean et al., 2016). As developers may freely try different requests, several times, and pose any query, the analysis of these logs gives insights about the usability of the WAPI.

Conversely, production logs are generated while applications are being used by end-users. Since the applications are released for final use, it is assumed that they are quite stable, thus the WAPI requests predetermined by the implemented functionalities of the applications are less error-prone. By analyzing these logs, providers may identify consumers' needs for new features, and implement the corresponding endpoints. These logs can reveal new usage scenarios providers may have not thought about, instructing them in including these scenarios in the documentation. Besides these, providers may identify ways of improving the WAPI based on how consumers use it, merging endpoints that are always called together for a specific purpose, removing endpoints

[1] http://httpd.apache.org/docs/current/mod/mod_log_config.html.

that are not being used or that are superseded by other endpoints, or creating new endpoints derived from the existing ones that are always called with specific values for some parameters (e.g., the new COVID-19 endpoint that Twitter released based on the high interest researchers had for the conversations on this topic[2]).

As the main goal of this work is to identify the need for changes based on the way consumers use the WAPIs in their applications, we make use of the production logs.

2.2. Process mining

Process mining is a process-oriented data mining discipline that uses event logs to extract process-related insights like building the process model, detecting discrepancies between the model and the monitored behavior, or improving the model based on the monitored behavior. We give the definition of the fundamental concepts of process mining, as presented in Van Der Aalst (2016), and adapt them to the WAPI domain:

- *Activity* - a specific step in the process. For WAPIs, the set of activities is the set of WAPI endpoints and consumers' actions (i.e., request methods like GET, POST, PUT to the endpoint).
- *Event* - an activity occurrence at a specific time. We refer to WAPI requests as events and identify them by activity names and timestamps.
- *Case* - a process instance. We refer as a case to a set of requests that an application is submitting to the WAPI during a certain time period. In the web domain, a case is commonly referred to as a session. Each event belongs to a specific case.
- *Event log* - a set of cases. In the WAPI domain, the event log corresponds to the WAPI usage log, which contains all the requests (i.e., events) that consumers (i.e., applications) make against the WAPI.

In order to apply process mining, the WAPI requests in the usage log should have at least three attributes: (i) case identifier that uniquely identifies the case to which each event belongs (composed from one or more columns), (ii) activity name that characterizes each event, and (iii) timestamp that indicates the time when an event occurs. The presence of these three attributes allows us to infer process insights from the event log. Certainly, other attributes that might store additional information about the events (e.g., application ID, status code), whenever available, add value to the analysis.

Typically, the output of process discovery is a process model, which can be a process map, a graph-based model, or other representations that describe the actual process based on the generated logs. In the WAPI context, process discovery consists on mining the usage logs to create the process model, where nodes represent the endpoints being called and directed edges the calling sequence of two endpoints. However, WAPIs are a typical example of less structured processes (they do not define a strict sequence of execution, but rather offer interactive functionalities like sharing data or passing messages Günther and der Aalst, 2007): consumers access the resources according to their needs, without following a strictly defined sequence of calls. As a result, the discovered process models turn out too complex, and consumers' behavioral patterns can hardly be identified. Even though there exist several techniques that tackle the simplification of complex models (i.e., clusterization, prioritization of activities and paths using several metrics as fitness and precision, or significance and correlation as in fuzzy miner algorithm Pedro et al., 2016; Günther and der Aalst, 2007), these kinds of simplifications may affect the accuracy of the identified behavior. As an example, we can mention the sequences with direct consecutive calls that can be found in the simplified process models. These sequences heavily rely on the level of simplification applied during the model creation. Hence, sometimes, the direct consecutive calls sequences shown do not really exist as the less frequently called activity in the middle of two more frequently called activities may be removed. To avoid identifying inaccurate patterns, we are not opting for any simplification of the complex spaghetti-like process. In fact, the complexity of the end-to-end process model does not limit, nor strongly affect our approach, because we are interested in the narrow relationships between a limited number of endpoints (2 to 4 endpoints), likely called together to perform a specific task. Therefore, we focus on the zoomed-in views of the end-to-end process model and put particular attention and interest in how specific nodes are related to each other.

2.3. Graphlets

To find all the possible ways the endpoints can be connected with each other and to characterize the structural relationships between them, we make use of the concept of graphlet. Graphlets represent small, connected, and non-isomorphic sub-graphs used to characterize and compare the structure of larger networks by detecting their local structure properties (Pržulj et al., 2004). There exist several graphlet-based methods that exploit the information encoded in large networks by counting all the graphlets of these networks. Generally, graphlet-based methods for analyzing directed networks, as in our case, do not consider graphlets with more than four nodes, due to the composed nature of their structure (i.e., subsumption of smaller graphlets) and also for computational reasons (Sarajlić et al., 2016).

Fig. 1 depicts the set of all possible directed graphlets composed of up to four nodes. Nodes represent WAPI endpoints, while edges represent their calling order: an edge from node A to node B shows that endpoint B was called after endpoint A. The general notion of directed graphlet, as defined in Sarajlić et al. (2016), includes graphlets from two to four nodes, and does not contain the ones that connect two nodes with edges in opposite directions. In order to represent the possible consumers' behaviors, we added the graphlets $G1$ and $G2$ representing a graphlet with only one node, and $G4$ as the graphlet with edges in opposite directions.

3. Approach

In this section, we give a general overview of our approach and the main steps we follow to identify the need for potential changes from the WAPI usage logs. It allows analyzing the usage logs in an iterative fashion. Thus, after implementing the detected changes, providers can monitor how consumers are adapting to them. Then, based on consumers' new behavior (reflected on the newly generated usage logs), providers may introduce new changes.

As our main objective consists in suggesting WAPI changes based on the extracted behavioral patterns, we need to specify beforehand which specific behaviors may reveal different consumers' needs or may be an indication for change. Indeed, instead of blindly searching for any usage patterns, we prune the search space to the patterns expressing consumers' needs for change. Therefore, different from techniques that mine the process model and extract from it frequent patterns (Bose and der Aalst, 2009; Leemans and der Aalst, 2014; Chapela-Campa et al., 2019), in our approach, we pre-define the set of patterns for which we look into the logs.

[2] https://twittercommunity.com/t/new-covid-19-stream-endpoint-available-in-twitter-developer-labs/135540.

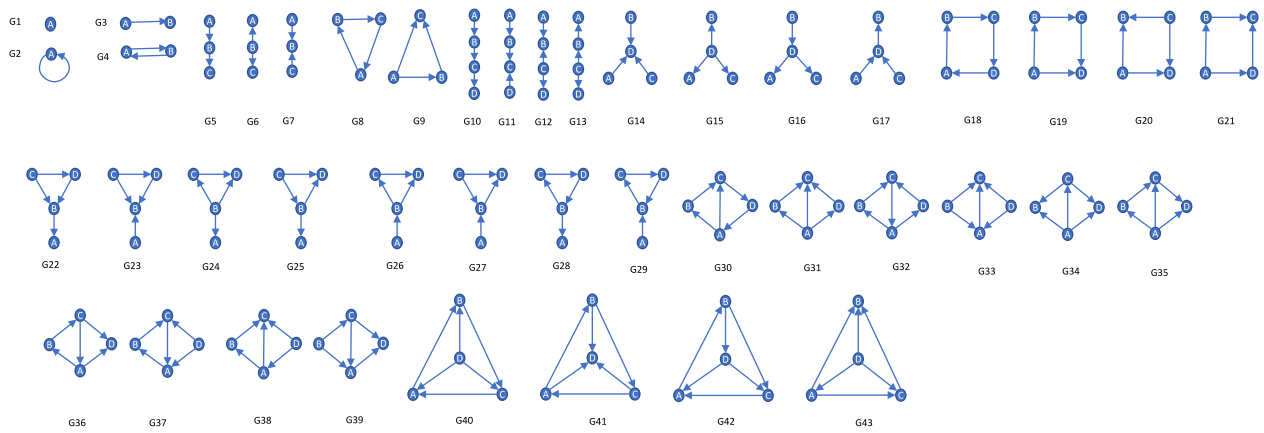

Fig. 1. Directed graphlets with up to 4 nodes (Sarajlić et al., 2016).

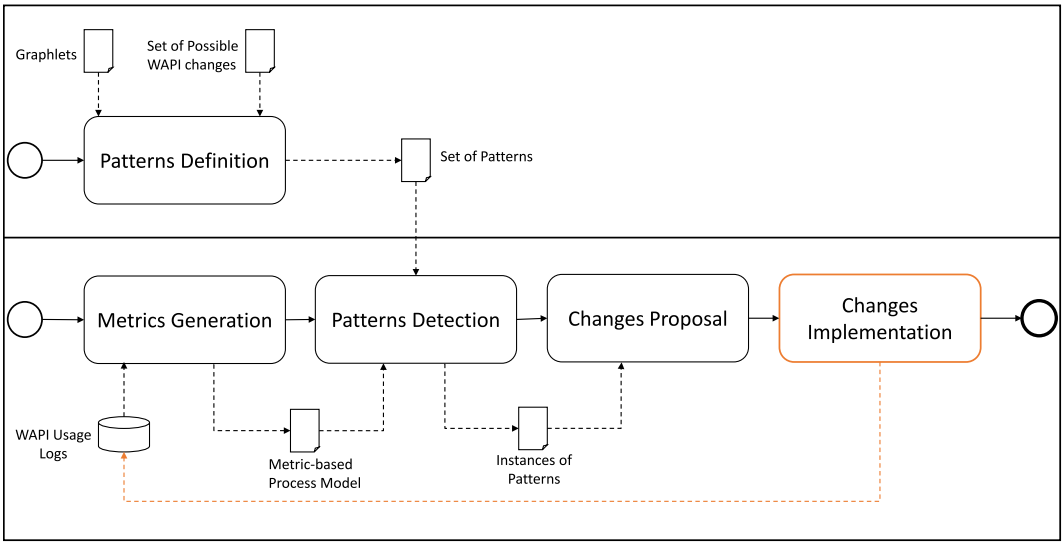

Fig. 2. Main steps of the approach. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 2 depicts the main steps of the approach, which consists in two crucial stages. The first stage (illustrated at the upper part of the figure), which comprises the *'Patterns Definition'* step, can be performed every time providers want to detect different users' behavior through specific patterns. If providers are interested in following other types of behavior (not only those that indicate the need for change), they can include them in this first step, and then proceed with the other steps accordingly. In our case, we search the graphlets that encode structural relationships between endpoints representing changes that can happen to WAPIs according to Koçi et al. (2019), to define this way the set of patterns. We explain this step in more detail in Section 4.

The second stage (illustrated at the bottom part of the figure) can be performed multiple times, periodically when more usage logs are generated. During *'Metrics Generation'* step, we analyze the WAPI usage logs using process mining, build the order in which the consumers call the WAPI endpoints and, for each endpoint, we calculate several metrics, to obtain a metric-based process model. Having the set of patterns and the metric-based process model, we detect the occurrences of patterns in the usage log (*'Patterns Detection'*). We associate a change to each pattern and suggest them to the providers (*'Changes Proposal'*). Although some WAPI usages may be interesting but no frequent, we focus on those usages that are frequent, as the changes they imply would bring more improvements. As one endpoint might appear in several patterns, domain knowledge should be used to prioritize the patterns and evaluate the feasibility of changes' implementation. WAPI providers can properly evaluate the patterns, and decide if the indicated changes can be implemented (taking into account not only consumers' usage but also business requirements or maintainability issues, which cannot be observed from the logs). If providers decide to implement the changes (*'Changes Implementation'* is an external step done by providers, thus shown in orange in Fig. 2), consumers have to use the new WAPI endpoints. The logs generated after the consumers adapt to the new changes will show if consumers are using the WAPI endpoints as expected. Even though analyzing the new-generated logs is out of the scope of this paper (shown with the orange arrow in Fig. 2), it poses opportunities to not only review consumers' adaptation to the changes but also to measure the improvement brought by the changes. This indicates that our approach seamlessly works whether it is its first iteration or it is being applied after implementing some of the previously proposed changes. If introduced in the development lifecycle of WAPI, our approach can eventually converge if the usage is stable/constant, but it can also react and detect occasional changes in the behavior (new data requirements, new types of users/applications, etc.) and propose an adaptation of the WAPI to this new behavior.

In summary, the study presented in this paper answers the following research questions:

RQ1: In which ways can endpoints be related to each other to indicate the need for change based on how consumers call them? We extract from the given graphlets all the possible ways endpoints can be connected and then summarize in a set of behavioral patterns those relationships whose occurrences in the logs may indicate room for improvements or suggest the need for specific WAPI changes.

RQ2: What information can we extract from usage logs, that can help in detecting the defined relationships between endpoints? We introduce the set of metrics needed to identify the occurrence of the defined patterns. These metrics' values will be extracted and calculated from the usage logs.

RQ3: How can we detect the occurrence of the patterns in the usage logs? We expound the detection of the defined set of patterns, using the introduced metrics.

RQ4: To what extent can we propose changes based on the patterns found in the usage logs? After detecting the occurrence of the patterns in the logs, we evaluate their significance, first from consumers' point of view (i.e., to see if the patterns manifest consumers' need for improvements), and second from providers' point of view (i.e., to see if the changes we propose are feasible and beneficial for providers). Hence, we divide this question into two sub-questions:

RQ4.1: To what extent can we understand consumers' needs based on the behavior manifested in usage logs?

RQ4.2: To what extent can we interpret the identified consumers' needs into feasible future changes?

4. Defining and detecting the WAPI behavioral patterns

In this section, we describe in more detail the main steps of our approach. In Section 4.1 we answer RQ1 by defining the set of patterns. In Section 4.2 we answer RQ2 by introducing and describing the metrics we generate after applying process mining. Then, in Section 4.3 we answer RQ3 by exhaustively explaining the detection of the patterns. We answer RQ4 later in Section 5.

4.1. Patterns definition

Using process mining concepts, we give the following definitions:

Definition 1. A sequence is an ordered occurrence of activities within a case.

Definition 2. A pattern represents a relationship between activities and is inferred from frequent sequences that these activities are part of. Each pattern reveals a specific consumer behavior and might be an indication for change.

In order to define all the structural relationships the graphlets represent in terms of sequences in the logs, we extract all possible sequences that can be generated from each of the graphlets of Fig. 1. In our case, there are no well-defined first or last-called activities. Thus, nodes that have at least one outgoing edge can be the first called endpoints of the sequences, while nodes that have at least one incoming edge can be the last endpoints of the sequences. To get the longest sequences from each graphlet, we start from nodes with only outgoing edges and finish with nodes with only incoming edges (if available).

Then, for every two endpoints, we analyze the sequences they are part of. If consumers' behavior manifested in these sequences (e.g., repetitive calls, consecutive calls) indicates the need for a change (Koçi et al., 2019) (e.g., creating new parameters, merging endpoints), we classify the relationship as a pattern along with the change it indicates.

Even though the graphlets are built with unlabeled nodes (the nodes differentiate from each other by their position in the graphlet, meaning their incoming and outgoing edges), we are labeling the nodes to simplify the explanation of the generated sequences. It is important to note that even though we are interested in the relationships between at most two endpoints, other endpoints called by consumers close to these two play an essential role in identifying and detecting these relationships.

To illustrate how we elicit the patterns from the graphlets of Fig. 1, we give two extended examples:

- Let us look at graphlet G9 and list the sequences that it can generate. Node A has only outgoing edges, so this node will be the first called endpoint of the sequences. In the same way, node C has only incoming edges, therefore it will be the last called endpoint of the sequences. Thus, the sequences that this graphlet can generate are {(A, B, C), (A, C)}. Analyzing the relationship between endpoint A and B, the derived relationship from this graphlet is: an endpoint that is optional or complementary to another (i.e., B is optional between A and C). The same relationship can be found in G22, G23 (i.e., D is optional between C and B), etc. (more details in Appendix A).
- Let us look at graphlet G21 and list the sequences that it can generate. Node A has only outgoing edges, while node C has only incoming edges, meaning that the generated sequences will start from A, and end in C. Thus, the sequences that this graphlet can generate are {(A, B, C), (A, D, C)}. The derived relationship is: endpoints that are possible choices between other endpoints (i.e., B and D called between A and C). The same relationship can be found in G31, G32, etc. (more details in Appendix A).

Same relationships can be derived from different graphlets. We summarize the identified relationships into seven WAPI behavioral patterns, as in Fig. 3. It is important to note that one endpoint may be part of more than one pattern occurrence, but the prioritization of the changes that the patterns indicate will be made using experts' domain knowledge.

The implied changes target WAPI improvement in different aspects: (1) reduce the number of WAPI calls consumers have to make to get the needed data (and consequently reduce consumers' costs, in case consumers are paying-per-WAPI request) (P1, P2, P6), (2) decrease the traffic between applications and WAPI server (P1, P2, P6), (3) reduce the time needed to get the desired result from the WAPI (considering each request as an opportunity for consumers to make a mistake, and for WAPI provider to return an error message Medjaoui et al., 2018) (P1, P2, P3, P6), (4) simplify the WAPI by getting rid of endpoints whose functionalities can be combined (P2, P3, P4, P5, P6, P7), (5) reorganize endpoints that serve the same purpose and supersede each other (P3, P4, P5, P7).

4.2. Metrics generation

In this section, we respond to RQ2. We introduce a set of metrics, that can be quantified by extracting information from the logs.

To detect the occurrence of the patterns in the logs, we use two basic concepts from the association rules research field, support and confidence (Agrawal et al., 1993), and adapt them

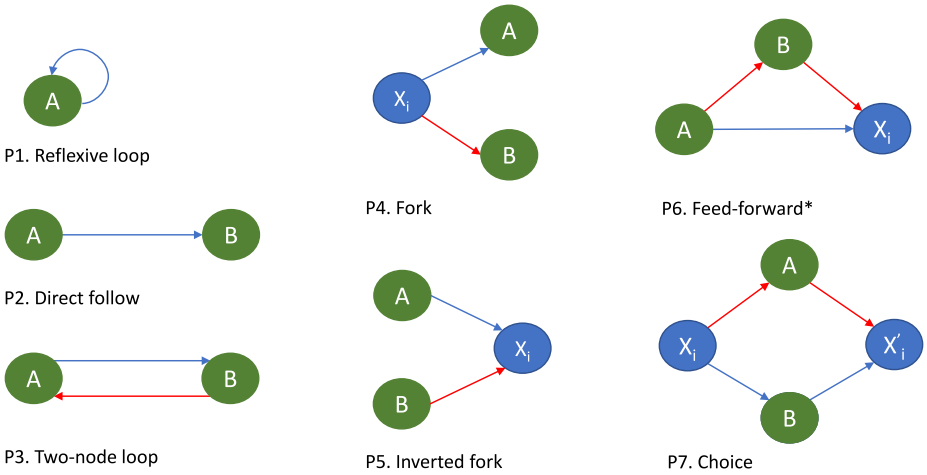

Fig. 3. The set of patterns. Edges with different colors belong to different cases. Green nodes represent the endpoints whose relationship the pattern describes. Blue nodes represent the endpoints that help detect the pattern. [∗] The name for this pattern is adopted from Milo et al. (2002), where the authors introduce several network motifs (specific sub-graphs) that appear frequently in different complex networks. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to the context of our approach. The *support* of an association rule of the form $X \Rightarrow Y$ is the fraction of instances where both the antecedent X and consequent Y hold. In our context, we refer as support to the frequency with which the instances of a pattern appear in the log file. For instance, given a pattern that describes a causal relationship between two endpoints (e.g., A being called implies B is called directly afterward), the support of this pattern will be equal to the frequency of the edge (A, B). For patterns formed with more than one sequence (e.g., A being called implies B can be called or C can be called), we take the geometric mean of the frequencies of the sequences that form it, (A, B) and (A, C) respectively. We do so to properly summarize the frequencies, and reduce the effect of skewed data (i.e., one of the two sequences has a considerably higher frequency than the other). We use absolute frequency (i.e., the total number of times that a sequence was called, not normalized by the size of the log file) as endpoints may be completely independent of each other, and the frequency of some endpoints with very high usage should not undermine the frequency of the other endpoints. Consequently, support values may not be in the same range.

The *confidence* of an association rule shows the fraction of instances containing the antecedent X, for which the rule $X \Rightarrow Y$ is satisfied. In our context, we refer as confidence to the number of times the pattern really occurs in the log, over all the possible times it could have occurred. We express it as the relative frequency of the sequences that form the pattern, regarding the frequency of the nodes whose relationship the pattern describes. For instance, to continue with the same example as in support, the confidence of the pattern describing the relationship between endpoints A and B (A being called implies B is called directly afterward), will be the frequency of the edge (A, B) (the number of times B is called directly after A), divided by the frequency of A (the number of times B could have been called after A). We subtract from the endpoint frequency the frequency of the endpoint self-loop (i.e., $freq^*_A = freq_A - freq_{AA}$), so this metric can better reflect the relationship endpoint A has with other endpoints. We use this pruned (from self-loops) frequency for all the patterns that describe the relationship of one endpoint with another endpoint, different from itself.

Typically, in process mining, for each node and edge, several frequency metrics are extracted from the logs (e.g., absolute frequency, case frequency, maximum repetition). As support refers to the frequency of the patterns, we adapt it for different frequencies, as follows:

- Case Support — the number of different cases in which a sequence (or an endpoint, i.e., a sequence of length one) appears. The case support of a pattern shows how the overall absolute frequency of the pattern is distributed in several cases. For instance, a pattern with a high absolute frequency but a low case frequency indicates that few cases are following the pattern many times.
- Average Case Repetition Support — the average number of repetitions of following a sequence within a case (the ratio of absolute frequency and case support).
- Maximal Case Repetition Support — the maximum number of repetitions of following a sequence within the same case.
- Application Support — the number of different applications that have called a sequence. It also reflects the number of applications that will be affected by the changes that the pattern indicates.

In general, one is interested on those patterns (rules) with interestingness metrics (e.g., confidence, support) above some minimum thresholds. As the values for these thresholds may depend on complexity and cleanness of the data, or even business requirements and objectives, they usually are set by domain knowledge holders. We choose not to arbitrarily decide on the values of these thresholds (which would allow us to present the precision and recall of the mined patterns), as it might affect the soundness of the results. Instead, we make a more qualitative evaluation of the patterns.

4.3. Patterns detection

In the following of this section, we respond to RQ3. We describe each of the patterns, exemplify them to facilitate their understanding, and explain how we detect them in the logs.

Reflexive loops (identified in graphlet G2 in Fig. 1.) *Motivation:* The structure appears in the log as an endpoint called consecutively several times by a consumer, presumably with different values for the same attribute. It may point out a weakness in the endpoint design, that forces consumers to make unnecessary repetitive calls. Thus, it may indicate the need for a new parameter to which multiple values can be passed.

Example. Let us suppose that `GET api/events?date="15-12-2021"` returns all the events happening in `15 December 2021`. If consumers submit consecutive calls for different dates, then a

useful optimization that reduces the number of WAPI calls is to change the parameter `date` (or to create a new one), so that an array or a range of values can be passed.

Detection: We calculate the metrics for the pattern as follows:

$$support = freq_{AA}, \qquad confidence = \frac{support}{freq_A}$$

In addition to these two metrics, for the reflexive loop pattern, we calculate the average case repetition support, the maximal case repetition support, and also the longest sequence of reflexive loops. For instance, let us suppose that for the reflexive loop of the endpoint `GET api/events?date="date_value"` we have the following figures: the reflexive loop appears on average three times per case (the sequence (A, A) is found on average three times in each case), maximum fifteen times in one case (the sequence (A, A) is found on average fifteen times in each case), and the longest sequence of reflexive loops is eight (endpoint A is being called nine times consecutively in at least one case). If the suggested change will be implemented, it will improve the WAPI by reducing the number of calls on average three calls per case (each of the three sequences of (A, A) will be replaced by one call of A) and on a maximum of fifteen calls per case (each of the fifteen sequences of (A, A) will be replaced by one call of A), by replacing two calls of the endpoint with just one. As the longest sequence of reflexive loops is eight (the endpoint called nine times sequentially), there will be cases where nine calls will be replaced with one.

Direct-follow nodes (identified in graphlets G3, G5, G10, etc., in Fig. 1.) *Motivation:* The structure appears in the log as two endpoints called consecutively after one another. This pattern may imply a causal dependency relationship between these two endpoints, which forces consumers to call them together in that specific order to fulfill a specific task. As such, we suggest the possibility of merging the two endpoints into one that combines the functionality of both of them. Based on the dependency between the two endpoints, we define two types of the direct-follow relationship: *(i) endpoint B called always after endpoint A,* meaning that endpoint B is the dependent one, and *(ii) endpoint A called always before endpoint B,* meaning that endpoint A is the dependent one. Both relationships are similar as they represent an immediate succession between endpoints. They differ only on the position of the dependent endpoint (the first called endpoint of the sequence, or the second one), and this is reflected in the way we detect the pattern and calculate its metrics.

Example. Let us suppose that `GET api/sales` endpoint returns the list of all the classrooms of the faculty building, and `GET api/sales/reserves` returns the list of the all the reservations of the classrooms. If we detect that several consumers frequently call these two endpoints one after the other (type (ii) of the direct follow the *sales* endpoint is followed by the *reserves* endpoint), then we assume that the information about the reservations complements the classrooms' information. Therefore, we propose the creation of a new endpoint, which will embody the functionalities of both endpoints.

Detection: We calculate the metrics for the type (i) of the pattern as follows:

$$support = freq_{AB}, \qquad confidence = \frac{support}{freq^*_B}$$

For the type (ii) of the direct-follow pattern, the metrics will be calculated as follows:

$$support = freq_{AB}, \qquad confidence = \frac{support}{freq^*_A}$$

As mentioned, the difference consists in the node, whose dependent relationship the pattern describes. In the first one, the dependent node is node B, thus the confidence measures how significant is the incoming edge (A, B) for node B. In the same way, the confidence for the type (ii) measures the significance of the outgoing edge (A, B) for node A, as A is the dependent node, after which only node B is being called. To obtain more insights regarding the occurrence of the pattern in the data, we calculate the other mentioned metrics (naming the average case repetition support and the maximal case repetition support).

Two-node loop (identified in graphlet G4 in Fig. 1.) *Motivation:* This structure appears in the logs as two direct follow patterns in opposite order, i.e. (A, B) and (B, A). It describes the relationship between two endpoints that are called together, but not necessarily in the same order. Thus, it might indicate that these two endpoints have combinable functions and the data returned by them might be reorganized to better fulfill consumers' needs.

Example. Let us suppose that `GET api/program/local?name="MIRI"` and `GET api/program/international?name="BDMA"` endpoints are being called one after the other, in different order. This behavior can be an indicator that separating the endpoints to get the data about the offered master programs (local master MIRI and international master BDMA), does not match consumers' needs, who want to know the information of several masters regardless of their condition. WAPI providers can decide to offer just one standardized endpoint for the master programs `GET api/masterprogram?name={MIRI, BDMA}` (i.e., merge the two endpoints), and create a new parameter by which consumers can filter the data according to their needs. It is worth noting that we are not questioning the design decision of having two separate endpoints instead of a unified one. But, if consumers of the WAPI access those endpoints in that specific way, we recommend the merge, as the changed WAPI will better fit its own consumers' needs.

Detection: We calculate the metrics for the pattern as follows:

$$support = \sqrt{freq_{AB} \cdot freq_{BA}}, \qquad confidence = \frac{support}{\sqrt{{freq_A}^* \cdot {freq_B}^*}}$$

Fork (identified in graphlets G6, G13, G15, etc., in Fig. 1.) *Motivation:* This structure describes the relationship between two endpoints that are called after the same set of endpoints. It appears in the log as sequences of calls that have as target one of the two related endpoints, and as source one from the common set of endpoints. This pattern implies that the two related endpoints might have highly similar functionalities or purposes. As in Fig. 3.4, after calling endpoints X_i, consumers call endpoint A or B. Providers may consider this usage as an indicator to reorganize endpoints A and B, and merge them into one single endpoint.

Example. Let us suppose that after calling endpoint X_1 `GET api/analytics`, consumers call endpoint `A GET api/charts` to get the data for visualizing them in a chart form, or endpoint `B GET api/reportTables`, to get the data for visualizing them in a tabular format (we are using Fig. 3 nodes' label to simplify the example). These two endpoints, `A GET api/charts` and `B GET api/reportTables` are also being called after X_2 `GET api/dimensions` and X_3 `GET api/indicators`. This way of using the charts and reportTables endpoints (i.e., calling them after the same set of endpoints), may indicate to providers that these two endpoints can indeed be combined into a unified endpoint `GET api/visualization` with a new parameter by which consumers can filter the data according to their needs.

Detection: We calculate the metrics for the pattern as follows: Let $X = \{X_1, \ldots, X_N\}$ be the set of all endpoints such as $(X_i, A), (X_i, B) \in L$, for $i = 1, \ldots, N$, where L is the log file:

$$support = \sum_{i=1}^{N} \sqrt{freq_{X_iA} \cdot freq_{X_iB}},$$

$$confidence = \frac{support}{\sqrt{freq_A{}^* \cdot freq_B{}^*}}$$

As in the two-node loop pattern, we take the geometric mean of the frequencies of the sequences from each X_i to A or B and sum them to calculate the occurrence of the pattern.

Inverted fork (identified in graphlets G7, G11, G14, etc., in Fig. 1.) *Motivation:* This structure describes the relationship between two endpoints that are called always before the same set of endpoints. It appears in the log as sequences of calls that have as sources one of the two related endpoints, and as target one from the common set of endpoints. As in the *Fork* pattern, this pattern implies that the two related endpoints might have highly similar functionalities or purposes. As in Fig. 3.5, after calling endpoint A or B, consumers call any of endpoints X_i. Providers may consider this usage as an indicator to reorganize endpoints A and B, and merge them into one single endpoint.

Example. Let us suppose that after calling the endpoint `A GET api/rooms`, WAPI consumers might call the endpoint X_1 `GET api/reservations` to get the reservation about the room, the endpoint X_2 `GET api/location` to get the location of the room, or the endpoint X_3 `GET api/map` to get the room's position on the map of the campus (we are using Fig. 3 nodes' label to simplify the example). The same three endpoints, X_1 `GET api/reservations`, X_2 `GET api/location`, and X_3 `GET api/map`, are being called after the endpoint `B GET api/laboratories`. As the rooms and laboratories endpoints have the same possible choices, providers might consider reorganizing the data that these endpoints return, differentiating the type of the space by a parameter.

Detection: We calculate the metrics as above: Let $X = \{X_1, \ldots, X_N\}$ be the set of all endpoints such as $(A, X_i), (B, X_i) \in L$, for $i = 1, \ldots, N$, where L is the log file:

$$support = \sum_{i=1}^{N} \sqrt{freq_{AX_i} \cdot freq_{BX_i}},$$

$$confidence = \frac{support}{\sqrt{freq_A{}^* \cdot freq_B{}^*}}$$

Feed-forward (identified in graphlets G9, G22, G23, etc., in Fig. 1.) *Motivation:* This structure appears in the log as two different sequences, where one is formed by three endpoints called in sequence (A, B, X), and the other sequence skips the second endpoint, hopping from the source node directly to the target node (A, X). This relationship may imply that the second endpoint's functionality (i.e., B) might be optional or supplementary for the first endpoint. Thus, providers might decide to join the functionality of the second endpoint to the first one, by creating a new parameter for it. To simplify the illustration of the pattern, we consider one endpoint X_1, but endpoints A and B may be both followed by more endpoints (i.e., X_i).

Example. Let us suppose that endpoint `A GET api/competences` returns the list of all the competencies required for any degree, endpoint `B GET api/competences_type` returns data about the type of the competences (e.g., technical competence, transversal competence), and endpoint X_1 `GET api/subjects` returns the data about the needed subjects for a degree (we are using Fig. 3 nodes' label to simplify the example). Let us suppose that after calling the endpoint for the required competencies for a degree some consumers call the endpoint for the needed subjects for the degree. On the other hand, some other consumers call the endpoint about the competencies type after getting the data about the competencies, and then the endpoint of the subjects. By including the information about the competencies type to the competencies endpoint, providers will not only enhance the functionality of the competencies endpoint, but at the same time, they will reduce the number of calls for the consumers who were calling it always with the competencies type endpoint. Even though providers' action might be the same as in other patterns (merging endpoints), the structure of the pattern and the way it appears in the log is different, thus its detection is different.

Detection: We calculate the metrics as follows: Let $X = \{X_1, \ldots, X_N\}$ be the set of all endpoints such as $(A, B, X_i), (A, X_i) \in L$, for $i = 1, \ldots, N$, where L is the log file:

$$support = \sum_{i=1}^{N} \sqrt{freq_{ABX_i} \cdot freq_{AX_i}}, \qquad confidence = \frac{support}{freq_A{}^*}$$

Choices (identified in graphlets G21, G31, G32, etc., in Fig. 1.) *Motivation:* This structure describes the relationship between two endpoints that are called always before and after the same set of endpoints. This structure appears in the log as sequences of calls of three endpoints, that differ by the second called endpoint: (X_i, A, X_i') and (X_i, B, X_i'). After calling endpoint X, consumers may call endpoint A or B, and then endpoint Y. This behavior implies that endpoints A and B, as called together with the same endpoints, might be substitutes, meaning sharing similar functionality, or alternatives to each other, meaning having different functions, but potentially cater to a similar purpose. Thus, this usage might be an indicator for providers to merge these two endpoints into a new one, that consumers can call with different parameter values depending on their needs.

Example. Let us suppose that after calling the WAPI endpoint X_1 `GET api/student/ ID1` to get data about a specific student, consumers call the endpoint `A GET api/student/ID1/fails` to get the subjects where the student has failed, and then the endpoint X_1' `GET api/student/ID1/credits` to get the data about the credits the student has (we are using Fig. 3 nodes' label to simplify the example). On the other hand, other consumers after calling X_1 `GET api/student/ID1`, may call the endpoint `B GET api/student/ID1/passes` to get the data about the subjects the student has already passed, and then the endpoint X_1' `GET api/student/ID1/credits`. In this case, the providers might decide to merge the endpoints about the failed and the passed subjects into the same endpoint and filter the status of the subjects for a specific student using a new parameter.

Detection: We calculate the metrics for the pattern as follows:

Let (X, X') be the set of all pairs of endpoints such as (X_i, A, X_i'), $(X_i, B, X_i') \in L$, for $i = 1, \ldots, M$, where M is the set size, and L is the log file:

$$support = \sum_{i=1}^{N} \sqrt{freq_{X_iAX_i'} \cdot freq_{X_iBX_i'}},$$

$$confidence = \frac{support}{\sqrt{freq_A{}^* \cdot freq_B{}^*}}$$

5. Evaluation

In this section, we present the evaluation of our approach. In Section 5.1, we first introduce the two use cases in which we apply the approach. We tackle log preparation and its analysis in Section 5.2. In Section 5.3 we present the evaluation results.

5.1. Use case

In order to demonstrate and evaluate the defined patterns in real-world examples, we analyze usage logs from two WAPIs, belonging to two different domains.

The first set of usage logs comes from the District Health Information Software 2 (DHIS2) WAPI. DHIS2 is an open-source, web-based platform to implement health management, developed and maintained by the software development group within the Health Information Systems Programme at the University of Oslo (UiO), Department of Informatics. It is used worldwide by various institutions and NGOs for data entry, data quality checks, and reporting. It has an open REST WAPI, used by more than 60 native applications (consumers). For the analysis, we use logs from two different DHIS2 instances: (1) the World Health Organization (WHO), in their Integrated Data Platform[3] (WIDP), which is used by several WHO departments for routine disease surveillance and country reporting; (2) Médecins Sans Fontières (MSF), used for field data collection and as a central repository for medical data. Logs from these instances (gathered from the instances' server side) are from the period February to September 2019, and the logs together contain approximately 2.5 million requests.

DHIS2 WAPI resources (e.g., data, documents, functionality) are exposed through WAPI endpoints (e.g., `api/dataValueSets`). Using these endpoints, consumers can access and manipulate data stored in the instance of DHIS2, data related to diseases cases (e.g., where a disease or infection spread, number or cases), organization units collecting the data (e.g., hospitals), etc. Consumers can interact with the WAPI using HTTP methods: call a GET request to retrieve a resource, a POST request to create one, PUT to update a resource, and DELETE to remove it, e.g., `GET api/charts`, `POST api/dataSets/ID/version`.

The second use case belongs to a WAPI offered by the Barcelona School of Informatics at the Polytechnic University of Catalonia (Facultat d'Informàtica de Barcelona, FIB, UPC). The FIB WAPI, developed by inLab research laboratory,[4] provides a set of endpoints for extracting data about departments, courses, exams, rooms reservations, etc. It is a read-only WAPI, meaning that consumers can only retrieve the data using GET requests, e.g., `GET v2/assignatures`, `GET /v2/lectures`. It is built under REST architecture and is mainly being used by the FIB website, monitoring systems, school news screens, and several applications created for academic purposes. The log data we are using (gathered from the server side) dates from October 2019 to January 2020, and the logs contain approximately 1.8 million requests.

5.2. Data analysis

As the data in the WAPI logs are complex (e.g., unstructured, high volume data) and somewhat noisy, the information needs to be prepared according to the requirements of process mining analysis that will be applied. Thus, we performed a pre-processing phase, which consists of four steps: (i) *data fusion* that involves the merging of log files generated from different sources, (ii) *data cleaning* which consists of removing irrelevant and noisy log entries from the files (e.g., cases containing requests with client-side errors), (iii) *data structuring* that includes user and session identification, and (iv) *data generalization* that consists of extracting general WAPI specifications from the requests in the log files (Tanasa and Trousse, 2004; Koçi et al., 2021a). We have addressed extensively the problem of WAPI usage logs pre-processing in one of our previous works (Koçi et al., 2021a).

After the pre-processing phase, each log entry should have a distinct timestamp (to be able to establish the order of calls), a clear activity name (to identify the relevant calls made), and should refer to one case (to determine the execution of the process). Thus, a (normalized) log entry should look like this: "`24 5e93393e93c7ea99ea 2019-10-17 10:31:53 GET v2/quadrimestres/ID/assignatures 200 1233`", containing an identifier for the corresponding session (case id), application identifier, timestamp, method and endpoint (activity name), status code, and object size returned, respectively. We used the session identifiers as case ids, the request time as timestamps, the endpoints and the method as activities, and the other attributes as resources.

Considering the interaction between consumers and WAPI as a process, we applied process mining to build a metric-based process model. Simply put, for each node and edge, we calculated several figures (e.g., absolute frequency, case frequency), as specified in Section 4.

With the calculated figures for the nodes and edges and the set of patterns, we then detected the occurrence of specific patterns in our data and computed for each of them the two metrics, confidence and support.

Table 1
Interviewed WAPI providers and consumers.

WAPI	Developers	Experience with WAPI
DHIS2	EST, 2 developers	Consumers: Use DHIS2 WAPI to build applications
	MSF, 3 developers	Consumers: Use DHIS2 WAPI to maintain already built applications
FIB	inLAb, 1 developer	Provider: Develop and maintain FIB WAPI

5.3. Feedback from programmers

In this section, we evaluate our approach by responding RQ4. We do this by providing the empirical evaluation of RQ1-3, which have been responded in the previous section.

After detecting several patterns in the usage logs of the two WAPIs in study, the next step is determining two major things: (i) the extent to which these patterns are manifesting in consumers' behavior and their needs for improvement (RQ 4.1), and (ii) the extent to which these patterns are a real indication for change, from providers' point of view (RQ 4.2).

For these reasons, we evaluate our approach in two directions. First, we interview DHIS2 WAPI consumers, i.e., developers that have used the WAPI and have incorporated it into their applications. With this evaluation, we aim to assess the effectiveness of our approach in understanding consumers' needs and identifying areas for improvement from the log analysis. Furthermore, we interview FIB WAPI providers, i.e., those who have designed, developed, and still maintain the WAPI's endpoints. We aim to assess from the providers' perspective whether the suggested changes are properly addressing the detected behavior. As providers are the ones equipped with the needed knowledge on the WAPIs' endpoints, resources, functionalities, and the structure of the data behind the WAPI, they can better evaluate

[3] https://www.who.int/tools/who-integrated-data-platform.
[4] https://inlab.fib.upc.edu/en/inlab-fib.

Table 2
Consumers' interviews' results.

Nr.	Pattern	Dev.	Endpoint A	Endpoint B	Sup.	Conf.	Feedback
1	P2	Dev.1	GET api/programIndicators	GET api/programRules	1080	0.70	⊕
2	P5	Dev.2	GET api/userGroups	GET api/userRoles	623	0.54	⊕
3	P4	Dev.3	GET api/reportTables/ID	GET api/charts/ID	27 537	0.59	⊕
4	P1	Dev.3	PUT api/events/ID/ID		125 375	0.64	⊕
5	P2	Dev.4	GET api/programRules	GET api/programRuleVariables	730	0.44	⊕
6	P1	Dev.4	GET api/programIndicators		6257	0.62	⊕
7	P1	Dev.5	GET api/programIndicators		6257	0.62	⊕
8	P1	Dev.5	PUT api/events/ID		125 375	0.64	⊕
9	P4	Dev.1	GET api/analytics	GET api/charts/ID	41 083	0.57	⊘
10	P7	Dev.4	GET api/userSettings	GET api/attributes	10 810	0.42	⊘
11	P5	Dev.5	GET api/userGroups	GET api/userRoles	623	0.54	⊘
12	P7	Dev.1	GET api/programs	GET api/schemas	838	0.07	⊗
13	P2	Dev.1	GET api/dataElementGroups/ID	GET api/categoryCombos/ID	195	0.76	⊗
14	P2	Dev.1	GET api/constants	GET api/programs	148	0.01	⊗
15	P6	Dev.2	GET api/attributes	GET api/userSettings	4057	0.27	⊗
16	P6	Dev.2	GET api/schemas/organisationUnit	GET api/attributes	2419	0.26	⊗
17	P2	Dev.2	GET api/dataElementGroups/ID	GET api/categoryCombos/ID	195	0.76	⊗
18	P1	Dev.2	POST api/dataValues		139 859	0.82	⊗
19	P6	Dev.3	GET api/enrollments	GET api/events	300	0.36	⊗
20	P2	Dev.3	GET api/dataElementGroups/ID	GET api/categoryCombos/ID	195	0.76	⊗
21	P2	Dev.3	GET api/programIndicators	GET api/programs	90	0.01	⊗
22	P6	Dev.3	GET api/trackedEntities	GET api/programs	36	0.35	⊗
23	P5	Dev.3	GET api/userGroups	GET api/userRoles	623	0.54	◯
24	P4	Dev.4	GET api/attributes	GET api/systemSettings	11 768	0.56	◯
25	P2	Dev.4	GET api/programIndicators	GET api/programRules	1080	0.70	◯
26	P7	Dev.5	GET api/dataElementGroups	GET api/dataElementOperands	111	0.05	◯
27	P5	Dev.5	GET api/organisationUnitLevels	GET api/dataApprovalLevels	4628	0.68	◯

the importance of these patterns. On the other hand, besides technical expertise, they know the environment where the WAPI is being developed and maintained, the organizations that are exposing the data/functionalities, their policies, and requirements. Thus, they can determine the feasibility and applicability of the suggested changes.

Table 1 provides an overview of the intervied developers. We performed the consumers' interviews online and the provider's interviews face-to-face. The interviews followed a semi-structured format and took approximately thirty minutes per person. The patterns we detected from analyzing the usage logs were the main input for designing the interview.

5.3.1. Interviews with WAPIs consumers - DHIS2 use case

As we were able to get usage data from two consumers of DHIS2 WAPI, namely WIDP and MSF (introduced when presenting the DHIS2 case), interviewing developers from them was the most reasonable option. The two first developers were from EyeSeeTea (EST), a consulting company that maintains WIDP servers, develops, debugs, maintains, and upgrades DHIS2 official applications, as well as builds their own applications. We interviewed also three in-house developers from MSF, who are mainly working on maintaining DHIS2 applications, and not building them from scratch.

Before conducting the interviews, we sent to all interviewees a short questionnaire with a few more general questions regarding their background, experience working with WAPI, and a list of endpoints from the respective WAPIs that appear in one or more of the detected patterns. The interviewees were asked to select from the list the endpoints they were more familiar with, so we could customize the interviews' questions to their knowledge about the WAPIs.

We started the interview with some general questions regarding the evolution of DHIS2 WAPI, to better understand how changes are perceived from consumers' point of view (the whole set of questions can be found in Appendix B).

In general, consumers asked for more detailed and updated documentation. They did not complain about the frequency of changes, nor the compatibility (most of the newly introduced changes are compatible with previous versions), but when even a minor change is introduced without being documented, they experience several issues with their applications. They brought the example of the `api/events` endpoint that was recently changed from version 2.37.7 to version 2.37.8, without being documented. The change affected the response: before, this endpoint used to return ten different properties, and after the change, it was returning half of them. The other properties were excluded from the WAPI response. In case developers wanted those properties to be included in the response payload, they had to specify them in the request query. As this change was not documented, developers had to notice it themselves and react afterward, so that their applications would not misbehave.

Regarding the communication with providers, all the interviewed consumers confirmed that the DHIS2 community of practice is very active and that providers have made available several communication channels for consumers (e.g., JIRA, mailing lists, Slack channels). Providers respond to most of the issues raised by consumers and fix the reported bugs. This means that, to some extent, providers take into account explicit consumers' needs when implementing changes.

Next, we made more specific questions and asked the respondents about the usage of specific endpoints, and their opinion on specific possible changes that could be performed on those endpoints. We showed them endpoints from significant patterns (i.e., with high confidence and support), as well as endpoints from non-significant patterns (i.e., with low significance or support), without giving any insights regarding the value of respective metrics. This way, we could assess the importance of the detected patterns, prove that we did not miss any important issues with endpoints, and avoid a possible bias (interviewees thinking that all patterns are significant).

On average, we introduced five patterns to each developer. To evaluate as many patterns as possible, we aimed to use different examples for each developer. But, considering the set of endpoints they were more familiar with (selected from the pre-interview questionnaire), we sometimes had to show different developers the same example. However, the answers for the same patterns turned out to be neither conflictual nor contradictory.

Table 3
Consumers' answers' categorization.

Symbol	Endpoint(s) are used as in the pattern	The change is relevant (There is room for improvement)
⊕	Y	Y
⊘	Y	N
⊘	N	Y
⊗	N	N
◯[a]	–	–

[a]Consumers do not use the endpoint(s) as in the pattern in their use cases.

Table 2 shows the results of the interviews. Table 3 shows the different categories where consumers' answers could fall.

Case 1: Endpoints used as in the pattern, and the change is relevant ⊕. In seven of the patterns (eight rows in the table as one of the pattern occurrences, nr. 6 and nr. 7, were shown to two different developers), the interviewed developers said that they use the endpoints as in the patterns and that the endpoints could be improved. In some cases, they welcomed the idea of merging the endpoints used together, and for the cases where the merging was not directly embraced (an expected behavior from consumers to not directly agree on a change, as it will be followed by changes in their applications), they admitted having some issues with those endpoints and pointed out the need for refactoring. For pattern occurrence nr. 3 (i.e., fork: `GET api/reportTables/ID` and `GET api/charts/ID`), the suggested change to merge these endpoints together was already implemented in the latest version of the DHIS2 WAPI. The new introduced endpoint `api/visualization` unifies both `api/charts` and `api/reportTables` endpoints. As explained in DHIS2 documentation,[5] the main idea behind this change was to enable consumers to have a unique and centralized endpoint that provides all types of charts and report tables as well as specific parameters and configuration for each type of visualization. As Dev.3 confirmed, this change actually simplified the WAPI, as both `api/charts` and `api/reportTables` endpoints were being used to represent the same information.

Case 2: Endpoints used as in the pattern but the change is not relevant ⊘. In pattern occurrences nr. 9, *Dev.1* responded that he uses the endpoints as described in the pattern, but he did not find relevant the change suggested by the pattern. This pattern had relatively high support and confidence, thus it was frequently found in the logs. The developer explained that one of the endpoints is used to extract metadata, and the other to extract data, and that is why merging them is not an option.

Case 3: Endpoints not used as in the pattern but the change is relevant ⊘. In pattern occurrence nr. 10, *Dev.4* said that even though he was not using the endpoints as in the pattern (choice pattern: `GET api/userSettings` and `GET api/attributes`), one of the endpoints (`GET api/userSettings`) could be improved. In pattern occurrence nr. 11, *Dev.5* said that, even though he does not use the endpoints as in the pattern, there is room for improving them. He admitted that the logic behind endpoint `GET api/userGroups` is too complex and not intuitive.

Case 4: Endpoints not used as in the pattern and the change not relevant ⊗. In nine of the pattern occurrences (eleven rows in the table as one of the pattern occurrences, nr. 13, nr. 17, and nr. 20, were shown to three different developers), the interviewed developers responded that they do not use the endpoints as in the patterns, and the suggested change was not relevant. In seven of the cases, the patterns had low support and low confidence, so their answers were the expected ones. But two of the patterns that got this answer had high confidence. We showed the direct follow pattern (`GET api/dataElementGroups/ID`, `GET api/categoryCombos/ID`) with confidence = 0.76, to three different developers and got the same answer: they do not use these endpoints one after the other, and their merging was not seen as beneficial from their point of view.

The other pattern occurrence with high confidence, and at the same time high support, that got the same answer was the reflexive loop of the endpoint `POST api/dataValues` (row nr. 18). The developer said that the endpoint was working fine, they did not have any issue with it, thus there was no room for improvement. Interestingly, this pattern (reflexive loop) is a case where some consumers may not experience any inconveniences with the endpoint (if network latency, i.e., ping time is low), thus no improvement by the change. They may call the endpoint in a loop, and each iteration may generate a WAPI request. In high latency situations, the delay coming from too many requests might be significant compared to one large request. Differently, *Dev.4* and *5* showed interest and approved our suggestion to change the endpoint `GET api/programIndicators` (reflexive loop, nr. 6 and nr. 7), so that they could call it one time to get the data they need, instead of making several consecutive calls.

Case 5: Endpoints not used as in the pattern in their use cases ◯. In five of the pattern occurrences, the interviewed developers said that in their use cases, they do not use the endpoints as in the patterns, but they could not say anything regarding the suggested change, as long as their usage of the endpoints was limited in few use cases. While one of the pattern occurrences had low support and confidence (nr. 26), the other four had relatively high support and high confidence. We happen to introduce two of these patterns (nr. 2, nr. 11, and nr. 23 `GET api/userGroups` and `GET api/userRoles`, and nr. 1 and nr. 25 `GET api/programIndicators` and `GET api/programRules`) to three different developers. For the first pattern occurrence (`GET api/userGroups` and `GET api/userRoles`) *Dev.2* and *5* said that the logic behind `GET api/userGroups` could be improved, and that it needs some refactorings. For the second occurrence (`GET api/programIndicators` and `GET api/programRules`), *Dev.1* said that as long as there are cases where he uses these endpoints together, changes could be applied to improve them. These are the only cases where the same pattern, got different (but not contradictory) answers from different developers.

5.3.2. Interviews with WAPIs providers - FIB use case

As already introduced in Section 5, FIB WAPI is a private WAPI, built to mainly serve the creation of the faculty web page and other related applications. Therefore, providers and consumers happen to be from the same development group. Actually, this is not an isolated case. Typically in banks, other financial institutions, or several companies, the development group creates a private (W)API to specifically serve their needs for the companies/institutions' applications. Thus, providers' decisions in applying changes in the WAPI will be directly affected by consumers' needs, expressed explicitly.

We interviewed the main developer of FIB WAPI (the whole set of questions can be found in Appendix B). He stated that in most cases, consumers' needs to create, change and maintain their applications were the starting point of a change in WAPI. When they need new data, not supported by the current endpoints of the WAPI, they create the respective endpoints for that data.

Table 4 shows the results of the interview. Table 5 shows the different categories where providers' answers could fall. The answers' categories in Table 5 are different from the ones in Table 3 because, from the providers' side instead of talking about the usage of endpoints (providers design, build, maintain, and

[5] https://docs.dhis2.org/en/home.html.

Table 4
Provider's interview results.

Nr.	Pattern	Endpoint A	Endpoint B	Sup.	Conf.	Feedback
1	P2	GET v2/assignatures/ID/guia	GET v2/competencies	68 942	0.74	⊕
2	P1	GET v2/competencies/categories		33 668	0.93	⊕
3	P2	GET v2/sales	GET v2/sales/reserves	157 647	0.71	⊗
4	P4	GET v2/quadrimestres/actual	GET v2/jo/assignatures	6013	0.69	⊗
5	P6	GET v2/assignatures/requisits	GET v2/competencies	149	0.07	⊗
6	P7	GET v2/jo/avisos	GET v2/noticies	497	0.01	⊗

Table 5
Providers' answers' categorization.

Symbol	The change is relevant	The change is feasible
⊕	Y	Y
⊘	Y	N
⊗	N	–

change the endpoints), we talk about the relevance of a suggested change, and its implementation feasibility.

The change is relevant and feasible ⊕. The developer showed interest in two of the introduced patterns' occurrences, both of which had high support and confidence. He explained the reasons behind the decision to design the endpoints in the current way. Even though he agreed that the endpoints could be improved by merging them or by adding attributes, he had decided to create the endpoints in that way because of their specific use in the applications.

The change is relevant, but it is not feasible ⊘. In our case, we did not get any response for this category. Providers may admit that the suggested change is relevant, but for other reasons (that cannot be observed in the logs e.g., security, privacy) the change may not be feasible and they cannot implement it.

The change is not relevant ⊗. The developer did not show interest in four of the patterns' occurrences. While two of them had low support and confidence (nr.5 and nr.6), the two others (nr.3 and nr.4) had high values for these metrics. While endpoints GET v2/quadrimestres/actual and GET v2/jo/assignatures (pattern occurrence nr.4) need different authorization level to be accessed (GET v2/jo/assignatures returns the subjects where a specific logged-in students is enrolled), endpoints GET v2/sales and GET v2/sales/reserves (pattern occurrence nr.3) should not be merged as their functions are well defined this way.

6. Discussion

In this section, we discuss interviews' answers and interpret the main findings. In the end, we summarize the threats to validity.

We were able to get very useful insight from the answers the consumers gave to the general questions of the interviews. While stressing the importance of updated documentation, developers admitted that they would even prefer the WAPI to offer several ways of doing the same task, as long as all the endpoints are well explained in the documentation. While this may give the impression of creating confusion or increasing the complexity of the WAPI, detailed documentation can mitigate this adverse effect, and even turn it into an advantage by increasing the flexibility for consumers. This feedback can be very valuable for providers when they decide on the changes they will implement on the WAPI. Some changes may cause an adverse effect on consumers (e.g., breaking their applications, increasing the complexity of WAPI). Knowing how to mitigate these effects, providers may be more relaxed when implementing the changes.

Furthermore, developers confirmed the availability of different communication channels between providers and consumers. Providers usually reacted to issues raised by consumers, making these channels very conducive. This means that, to some extent, providers take into account consumers' needs when implementing changes. Our approach will help them to apply a more in-depth analysis of consumers' needs, by gathering their feedback in a more straightforward, inexpensive, and scalable way.

From all the interviewees' answers to the specific questions, which are detailed in previous section, the ones that may be of more interest to be discussed here are those that do not accept the changes implied by patterns with high confidence and support and those that accept changes implied by patterns with low confidence and support. As we cannot arbitrarily decide on thresholds' values for the confidence and support (it may depend on how clean the logs are, or how complex the WAPI is) to later classify the patterns as significant or insignificant (and categorize them as false positive/negative or true positive/negative), we make a more qualitative discussion and generally review the most controversial answers.

Patterns that have high confidence ($\geq$ ***0.5***) ***and support, whose suggested changes are not welcomed by consumers or providers.*** Patterns fall in this category due to the presence of applications' end users' behavior in the WAPI logs. The way consumers' applications are built and the way their users interact with them, may create some frequent patterns in the WAPI usage logs, that do not indicate the need for change. For instance, if the users of an application access two features in a certain order, the WAPI endpoints incorporated in those features will appear in the logs as called together. This was the case with one of the patterns we showed to consumer developers. The pattern occurrence appeared with high confidence of 0.76, and relatively low (considering the margin of error due to the noise in the logs, and the size of the log file) support of 195. The developer confirmed he was not using together the endpoints (i.e., direct follow: `GET api/dataElementGroups/ID` and `GET api/categoryCombos/ID`). However, when he demonstrated us the consuming application, the features (i.e., tabs) using these endpoints were next to each other in the application interface, presumably guiding the end-users to access them one after the other. Thus, the behavior of the application's end-user was reflected in the logs.

Second, even though patterns may have high metrics' values, semantically they cannot be merged. This was the case of the endpoints `GET api/analytics` and `GET api/charts/ID`. The fork pattern appeared with high support (41 083) and relatively high confidence (0.57). However, one of the endpoints was used to get metadata, and the other to get data. Thus, their merging was off the table. This example is very interesting, as it implies the need to semantically check the patterns during pattern detection. Providers may exclude beforehand those patterns whose respective endpoints are built to extract different kinds of information (e.g., metadata and data).

The unavoidable presence of this group of patterns in the analysis results makes it necessary the evaluation of the patterns from the WAPI providers, that possess the needed domain knowledge to filter out the insignificant patterns. The blind evaluation based only on the value of metrics may result in wrong conclusions.

Patterns that have low confidence and/or support, but indicate the need for changes in the WAPI. Patterns can fall in this category if the dataset is not large enough. As one of the main aspects of our approach is the repetition of consumers' behavior, the lack of data may impede the discovery of interesting patterns. When showing non-significant patterns, developers did not show interest in any of them. They gave examples of using the endpoints independently of each other, thus not agreeing with the changes to merge the endpoints.

Threats to validity: Most of the internal validity threats concerned the pre-processing of the usage logs. As we mentioned in Section 5.2, the pre-processing of the usage logs was not trivial. We tackled some challenges when introducing the main steps of pre-processing. Each challenge can affect the validity of the results if the related issues are not identified and properly handled. We can mention here the session identification challenge. Most of the WAPIs are stateless, meaning that the server does not store the state, thus no sessions are generated. We constructed the sessions by assigning identifiers based on the 30 min timeout approximation (requests with a time difference of fewer than 30 min, coming from the same user, are part of the same session). This might add noisy sequences of calls (for applications with smaller timeout values) or exclude important ones (for applications with greater timeout values). To mitigate the risk of the 30-min approximation affecting the analysis, we computed several metrics for each pattern and the respective endpoints, so that the providers can have a more complete view of their usage.

One of the construct validity threats consists of misinterpreting the behavior of applications' end-users as applications' behavior toward the WAPI. The way end-users interact with the applications should not interfere with the analysis of the way applications consume the WAPI. To mitigate this threat (and to assess the effectiveness of our approach), we included in the loop the WAPI providers to properly evaluate the significance of the detected patterns.

Regarding internal validity, being aware of the possible researcher biases that might impact the interviewees' answers, we tried to be as neutral as possible during the interviews, paying particular attention to the questions' creation and the patterns' occurrences we introduced to the interviewees. Besides randomly selecting patterns' occurrences (taking into account the endpoints selected by interviewees in the questionnaire) with very diverse metrics' values, we chose not to show these values to the interviewees. Thus, their answers would not be influenced by the frequencies of the patterns, but only by their knowledge and experience with the respective WAPIs.

Furthermore, we were aware of the threat to validity concerning the subjective opinions interviewees may have regarding several endpoints. We mitigated it by showing to different interviewees same patterns and then observing their feedback. In none of the cases, their feedback was conflictual, or contradictory, affirming that they have assessed each pattern objectively based on their experience using the WAPIs.

To better understand the needs of WAPI consumers, one may have to include in the analysis the activities that consumers are performing after accessing the WAPI. As the WAPI is not used in isolation, its activities and the activities in the consumers' applications, outside the WAPI, are correlated. For instance, after accessing some data from the WAPI, an application may parse the data or cast the data type, and then use them according to its functionality. In order for the WAPI providers to identify the need for changing the data type, or shaping the data in a more useful way for their consumers, they should correlate and analyze both the WAPI usage logs and applications running logs. But, knowing the independent nature of WAPIs, how loosely coupled they are with their consumers (Pautasso and Wilde, 2009) and taking into account that consumers' applications code is not always public, analyzing these services together becomes unrealistic. As we analyze only the WAPI usage logs, our challenge is to understand consumers' behavior and identify their needs by making use of their footprints in the logs.

7. Related work

In this section, we review those works that are related to the intention behind (W)API evolution and the most followed practices by providers, as well as the use of process mining in analyzing WAPI behavioral patterns.

Granli et al. (2015) analyzed the driving forces of traditional API evolution using the existing software evolution theories, more specifically Lehman's laws. They showed that the two largest forces driving API evolution were the need for new functionalities and usability improvements. Their study revealed that business requirements and API consumers stand behind these forces, but they did not further elaborate on these factors. Sohan et al. (2015) investigated changes made when new WAPI versions were released, and how these changes were documented and communicated to consumers, without considering the reasons that lead providers in applying those changes. Several works are done in classifying the changes that often happen to WAPIs from older to newer versions (Koçi et al., 2019; Wang et al., 2014; Li et al., 2013). The addition of new elements (Koçi et al., 2019; Wang et al., 2014) and refactorings (Li et al., 2013) were the most common ones. Li et al. (2013) analyzed the API consumers' reaction toward the changes; however, they did not examine how the need for these changes was manifested in the consumers' behavior.

Regarding providers' practices when evolving their WAPIs, Espinha et al. (2014) investigated four well-known WAPIs (Google Maps, Facebook, Twitter, Netflix) and the evolution policies their providers follow. Their study showed that there is a lack of industry-standard regarding the evolution of WAPIs, and different providers follow different practices. However, the authors were more focused on the way providers introduce the new changes and communicate them to consumers, rather than on the origin of the changes. Lamothe et al. (2021) conducted a systematic review on API evolution literature. Besides the continuing change and growth, the increasing complexity and declining quality, and the conservation of familiarity (all considered worthy-challenged by the authors), they particularly stressed the need to leverage the feedback systems involved in API evolution. Mathijssen et al. (2020) performed a systematic literature review on different API management practices applied by providers. Access logging and usage monitoring were commonly followed by API providers, but with the aim of providing performance statistics or traffic metrics. Medjaoui et al. (2018) accentuate the need for proper and continuous API Management, by proposing guidelines to build, deploy, operationalize, refine, and evolve (W)APIs continually at scale.

There are few studies that consider consumers' behavior as feedback that should be taken into account when evolving a WAPI. Zibran et al. (2011) analyzed the issues raised by API consumers in five different bug repositories to identify the most reported API usability issues. Their study involved consumers, but not by analyzing the way they used the API, but the issues reported explicitly to the providers. Macvean et al. (2016) analyzed as well the usability of WAPIs. They took data from Google API Explorer to identify WAPIs, with which developers struggle more and spend more time and effort in learning and using. They were focused on the way developers perceive and learn the WAPI, and not how they were using endpoints based

on the functionality. On the same line, is one of our previous works (Koçi et al., 2020), where we analyzed development logs to measure the usability of WAPIs, to further suggest changes that would increase the perceived usability from consumers' point of view. Suter and Wittern (2015) analyzed WAPIs usage logs using trained classifiers in order to infer WAPI description. Even though with a goal different from ours, they pointed out the incomplete and noisy nature of WAPI usage data. They encourage future work not only on mitigating the quality issues of these data but also on broadening the goals spectrum of the usage logs analysis.

Related to the use of process mining in analyzing WAPI usage logs, we can find several works akin to our study in some particular aspects, like input data where the process mining is applied, not focusing on the end-to-end process but on part of it, etc.

Van Der Aalst (2012) and Van Der Aalst (2013) showed the potential of applicability of process mining in the services domain, identifying as well the main challenges like the correlation of the events. In both works, it was pointed out the need for additional research in improving the applicability of process mining in loosely coupled systems.

Günther and der Aalst (2007) focused on addressing the problem of the less structured processes (like accessing WAPI), where the discovered models are usually "spaghetti-like", not understandable, and uninformative. They presented a new process mining approach to overcome this problem by simplifying the model using two metrics: significance and correlation. Even though they state that their approach is universally applicable, we do not opt for any simplifications as we are interested in the direct order of WAPI calls and simplifications may affect the accuracy of patterns.

Zhong et al. (2009) have developed a framework for automatically mining API usage patterns (MAPO). Different from us, they mine applications' code snippets (no usage logs), focus on traditional API (no web API), and recommend usage patterns for consumers to better use the API (no for providers to evolve the API based on the implied changes). Wang et al. (2013) claim to outperform MAPO by introducing UP-Miner approach. They introduce two quality metrics, succinctness and coverage, that help to mine better API patterns (not redundant, more practical, more qualitative), so that developers (API consumers) would adopt them more easily. The same goal drove Nguyen and Nguyen (2015). The authors introduce a graph-based statistical language that analyzes applications' source code to suggest accurate API usage patterns. However, same as Zhong et al. (2009) and Wang et al. (2013), they focus on traditional API and mine applications' code.

Leemans and der Aalst (2014) introduced a technique to discover frequent episodes (i.e., partially ordered set of events) in event logs, using the notion of frequent itemsets and association rules. Their work, like ours, is somehow positioned between process mining and pattern mining, but unlike us, they do not pre-define any patterns but look for the most frequent ones. Bose and der Aalst (2009) focuses on discovering common patterns in process execution traces. They pre-define a few patterns that capture the manifestation of commonly used process construct and propose an approach to form an abstraction of activities using these patterns. Even though the presented patterns could be applied in several domains, we had to define our own set of patterns, which are not only specific to the WAPI case but also capture specific consumers' behavior and imply WAPI change.

Even though the above-mentioned works differ from ours regarding the input of the approach, the goal, or the methods followed, they helped us identify the current gaps in WAPI evolution practices. Reviewing those works helped us gain the right perspective on WAPI evolution and the application of process mining in our use cases. We identify a set of usage patterns, by analyzing usage logs, thus avoiding the analysis of applications' source code (not always available). We associated each pattern with potential changes in WAPI and presented their detection in the logs, opening new opportunities for providers to differently evolve their WAPIs (driven by consumers' behavior).

8. Conclusion and future work

In this paper, we presented an approach on how the analysis of WAPI usage logs can help providers to better understand the consumers' needs for specific improvements. Typically, WAPI providers log all the requests against the WAPI. Consumers' behavior is recorded on these usage log files. Hence, providers can infer useful information from their analysis. We defined seven behavioral patterns, whose occurrence can indicate the need for potential changes on the WAPI. We observed the occurrence of these patterns in two real-world examples and suggested potential changes that could improve the WAPI from the consumers' perspective. The feedback from consumers and providers of the WAPIs in study confirmed the promising potential of our approach. Moreover, recently, DHIS2 WAPI providers had been introducing few changes, motivated by the same ideas and goal as in our approach: to enable consumers in having well-defined and simplified endpoints, that would ease their experience with the WAPI.

In our future work, we aim to define more patterns, that cover more usage scenarios and capture more complex consumers' behavior. In order to evaluate the effectiveness of the proposed patterns and changes for the consumers, we plan to analyze the WAPI usage logs after the implementation of these changes, to see if the consumers are using the WAPI in the way we thought they will. Moreover, we intend to focus on the creation of useful and relevant migration scripts for the changes we suggest.

We plan to perform an additional evaluation to assess the providers' benefits coming from applying our approach during their WAPI evolution. Besides evaluating the benefits, a very interesting direction to expand the work would be to analyze and evaluate the providers' maintainability burden that specific changes may bring. Studying the break-even point between consumers' retention and satisfaction and the maintenance burden caused by the addition of some functionality would be very promising, as currently, it represents a research gap. Another way this work can be expanded is by studying the WAPI evolution from the business point of view: analyzing the companies' requirements that drive several changes. This way, the picture of the intent behind the changes will be more complete, and the prioritization of the changes coming from both consumers and companies can be somehow automated.

CRediT authorship contribution statement

Rediana Koçi: Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Validation, Writing - original draft, Visualization. **Xavier Franch:** Conceptualization, Methodology, Formal analysis, Investigation, Writing – review & editing, Supervision, Funding acquisition. **Petar Jovanovic:** Conceptualization, Methodology, Formal analysis, Investigation, Resources, Validation, Writing – review & editing, Supervision. **Alberto Abelló:** Conceptualization, Methodology, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

This paper has been funded by the Spanish Ministerio de Ciencia e Innovación under project/funding scheme PID2020-117191RB-I00/AEI/10.13039/501100011033. We especially thank Dr. Pedro Albajar-Viñas from WHO, who provided us the access to the DHIS2 WAPI logs, Jaume Moral from inLAb, who provided us the access to the FIB WAPI logs, and the developers from EST, MSF, and inLab for participating in the interviews.

Appendix A. Structural relationships derived from graphlets

See Table A.6.

Appendix B. Interviews

Interview with DHIS2 WAPIs' consumers

Q1: Based on your experience using DHIS2 WAPI, share your opinion on its evolution (frequency, changes relevance, or practices followed by providers).
Q2: Do you communicate with WAPI providers? If yes, how?
Q3: Do providers take into account your requests when changing the WAPI?
Q4: Would you prefer having different ways of doing a task (or get some data), or just one way?
Q5: Have you ever used the following endpoints together when implementing a specific feature in your applications? (direct-follow, two-node loop, and feed-forward)
Q6: If it was available a unified endpoint that combines the two mentioned endpoints, would you use it, or would you continue using the individual ones? Why?
Q7: Can you give any examples on how you use the following endpoints? (examples from fork, inverted fork, and choice)
Q8: How would you react to the implementation of a new endpoint that would combine the two mentioned endpoints? Would you be open to use it?
Q9: Have you ever felt the need to make several calls of this endpoint in series, with different values for its attributes or parameters?
Q10: Do you think there may be a more efficient way to get/post the needed data only by making one call, instead of several ones?

Interview with FIB WAPIs' providers

Q1: How do you deal with the evolution of FIB WAPI? What is the starting point of a change?
Q2: How do you communicate with your consumers?
Q3: Do you take into account the feedback from consumers when planning the changes?

Table A.6
Graphlets, the sequences they can generate, the spotted relationships between two nodes, and the inferred patterns.

Graphlets	Sequences	Relationships between nodes	Patterns
G1	A	Not connected node	–
G2	A, …, A	A node called consecutively k times	P1
G3	AB	Nodes called consecutively after one another	P2
G4	AB, BA, ABA	Nodes called in opposite direction	P3
G5	ABC	Nodes called consecutively after one another	P2
G6	BA, BC	Nodes that follow the same source node(s)	P4
G7	AB, CB	Nodes that are followed by the same target node(s)	P5
G8	ABCA, ABC, CA	Nodes called consecutively after one another Nodes called in opposite direction	P2, P3
G9	AC, ABC	A node being optional to another	P6
G10	ABCD	Nodes called consecutively after one another	P2
G11	ABC, DC	Nodes that are followed by the same target node(s)	P5
G12	AB, CD, CB	Nodes that follow the same source node(s) Nodes that are followed by the same target node(s)	P4, P5
G13	CD, CBA	Nodes that follow the same source node(s)	P4
G14	AD, BD, CD	Nodes that are followed by the same target node(s)	P5
G15	DB, DC, DA	Nodes that follow the same source node(s)	P4
G16	BDA, BDC	Nodes that follow the same source node(s)	P4
G17	ADB, CDB	Nodes that are followed by the same target node(s)	P5
G18	ABCDA, ABCD, DA	Nodes called consecutively after one another Nodes called in opposite direction	P2, P3
G19	ABCD, AD	A node being optional to another	P6
G20	AB, AD, CB, CD	Nodes that follow the same source node(s)	P4
G21	ABC, ADC	Nodes as possible P7s after another node	P7
G22	CBA, CDBA	A node being optional to another	P6
G23	CB, CDB, AB	A node being optional to another Nodes called consecutively after one another	P6, P2
G24	BA, BCD, BD	A node being optional to another Nodes called consecutively after one another	P6, P2
G25	CBA, CD, CBD	A node being optional to another Nodes called consecutively after one another	P6, P2
G26	ABCD, ABD	A node being optional to another	P6
G27	ABD, CD, CBD	A node being optional to another Nodes called consecutively after one another	P6, P2
G28	BCDBA, BCD, DB	Nodes called consecutively after one another Nodes called in opposite direction	P2, P3
G29	ABCDB, BCD, DB	Nodes called consecutively after one another Nodes called in opposite direction	P2, P3
G30	ABCDA, AC	Nodes called consecutively after one another A node being optional to another	P2, P6
G31	ABC, ACB, ADC	A node being optional to another Nodes as possible P7s after another node	P6, P7
G32	ABC, ADC, CA	Nodes as possible P7s after another node	P7
G33	BC, BAC, DC, DAC	A node being optional to another Nodes that follow the same source node(s)	P6, P4
G34	AB, AD, ACB, ACD	A node being optional to another	P6
G35	ABCD, AD, ACD	A node being optional to another	P6
G36	ABCD, ABCA, AD	Nodes called consecutively after one another A node being optional to another Nodes called in opposite direction	P2, P6, P3
G37	DABC, DC, ABCA	A node being optional to another	P6
G38	DC, DAC, DABC	A node being optional to another	P6
G39	BAD, BCD, BCAD	A node being optional to another Nodes as possible P7s after another node	P7, P6
G40	DBCAB, DCABC, DABCA	A node being optional to another Nodes that follow the same source node(s)	P6, P4
G41	ABCAD, ABD, ABCD	Nodes called in opposite direction	P3
G42	BC, BDC	A node being optional to another	P6
G43	DB, DAB, DACB, DCB	A node being optional to another Nodes that follow the same source node(s)	P6, P4

Q4: Are the following endpoints related in any way (e.g. usage, function, purpose)? (example from direct follow)
Q5: If consumers would call the previous endpoints together most of the time (one after the other always in the same order), would you consider this usage as an indication that there is a causal dependency relation between the two endpoints? Would this kind of relation encourage you to perform some action in order to improve these endpoints so that consumers can get the data more efficiently? Why yes/no?
Q6: Are the following endpoints related in any way? (example from the two node loop)
Q7: If consumers would call the previous endpoints together most of the times, without following a specific order, would you consider this usage as an indication that these endpoints have combinable functions? Would this kind of relation encourage you to perform some action in order to improve these endpoints so that consumers can get the data more efficiently and simpler? Why yes/no?
Q8: Are the following endpoints related in any way? (examples from fork, inverted fork, and choice)
Q9: If most of the times consumers would call the previous endpoints after and/or the same set of endpoints, would you consider this usage as an indication that these endpoints might have similar purpose, or joinable functionalities? Would this kind of relation encourage you to reorganize these endpoints to simplify the data retrieval for consumers, by providing endpoints without duplicate functionality? Why yes/no?
Q10: Are the following endpoints related in any way? (example from the feed forward)
Q11: If most of the times consumers would call the second endpoint optionally after the first endpoint, would you consider this usage as an indication that the second endpoint functionality might be supplementary or optional to the first one? Would this kind of relation encourage you to improve any of these endpoints so that consumers can get the data needed data more efficiently? Why yes/no?
Q12: If you see that consumers would call the following endpoint several time in sequence with different parameters' values, would you consider this usage as an indication that there is a need to improve this endpoints, so that consumers can get the same data with only one call? (example from the reflexive loop)

References

Abelló, Alberto, Ayala, Claudia, Farré, Carles, Gómez, Cristina, Oriol, Marc, Romero, Óscar, 2017. A data-driven approach to improve the process of data-intensive API creation and evolution. In: Proceedings of the 29th InternationalConference on Advanced Information Systems Engineering (CAiSE2017). CEUR-WS. org, pp. 1–8.

Agrawal, Rakesh, Imieliński, Tomasz, Swami, Arun, 1993. Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. pp. 207–216.

Bose, RP Jagadeesh Chandra, der Aalst, Wil MP Van, 2009. Abstractions in process mining: A taxonomy of patterns. In: International Conference on Business Process Management. Springer, Berlin, Heidelberg, pp. 159–175.

Chapela-Campa, David, Mucientes, Manuel, Lama, Manuel, 2019. Mining frequent patterns in process models. Inform. Sci. 472, 235–257.

Doerrfeld, Bill, 2018. 10+ API Monitoring Tools. Available: https://nordicapis.com/10-api-monitoring-tools.

Espinha, Tiago, Zaidman, Andy, Gross, Hans-Gerhard, 2014. Gross Web API growing pains: Stories from client developers and their code. In: 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE, pp. 84–93.

Espinha, Tiago, Zaidman, Andy, Gross, Hans-Gerhard, 2015. Web API growing pains: Loosely coupled yet strongly tied. J. Syst. Softw. 100, 27–43.

Granli, William, Burchell, John, Hammouda, Imed, Knauss, Eric, 2015. The driving forces of API evolution. In: Proceedings of the 14th International Workshop on Principles of Software Evolution.

Günther, Christian W., der Aalst, Wil MP van, 2007. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: International Conference on Business Process Management. Springer, Berlin, Heidelberg, pp. 328–343.

Koçi, Rediana, Franch, Xavier, Jovanovic, Petar, Abelló, Alberto, 2019. Classification of changes in API evolution. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference. EDOC, IEEE, pp. 243–249.

Koçi, Rediana, Franch, Xavier, Jovanovic, Petar, Abelló, Alberto, 2020. A data-driven approach to measure the usability of web APIs. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 64–71.

Koçi, Rediana, Franch, Xavier, Jovanovic, Petar, Abelló, Alberto, 2021a. Improving web API usage logging. In: Research Challenges in Information Science. RCIS.

Koçi, Rediana, Franch, Xavier, Jovanovic, Petar, Abelló, Alberto, 2021b. PatternLens: Inferring evolutive patterns from web API usage logs. In: International Conference on Advanced Information Systems Engineering. Springer, Cham, pp. 146–153.

Lamothe, Maxime, Guéhéneuc, Yann-Gaël, Shang, Weiyi, 2021. A systematic review of API evolution literature. ACM Comput. Surv. 54 (8), 1–36.

Lamothe, Maxime, Shang, Weiyi, 2020. When apis are intentionally bypassed: An exploratory study of api workarounds. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering. ICSE, IEEE, pp. 912–924.

Leemans, Maikel, der Aalst, Wil MP van, 2014. Discovery of frequent episodes in event logs. In: International Symposium on Data-Driven Process Discovery and Analysis. Springer, Cham.

Li, Jun, Xiong, Yingfei, Liu, Xuanzhe, Zhang, Lu, 2013. How does web service API evolution affect clients? In: 2013 IEEE 20th International Conference on Web Services. IEEE, pp. 300–307.

Macvean, Andrew, Church, Luke, Daughtry, John, Citro, Craig, 2016. API usability at scale. In: PPIG.

Mathijssen, Max, Overeem, Michiel, Jansen, Slinger, 2020. Identification of Practices and Capabilities in API Management: A Systematic Literature Review. [Preprint] https://arxiv.org/abs/2006.10481.

Medjaoui, Mehdi, Wilde, Erik, Mitra, Ronnie, Amundsen, Mike, 2018. Continuous API Management: Making the Right Decisions in an Evolving Landscape. O'Reilly Media.

Milo, Ron, Shen-Orr, Shai, Itzkovitz, Shalev, Kashtan, Nadav, Chklovskii, Dmitri, Alon, Uri, 2002. Network motifs: simple building blocks of complex networks. Science 298 (5594), 824–827.

Murphy, Lauren, Kery, Mary Beth, Alliyu, Oluwatosin, Macvean, Andrew, Myers, Brad A., 2018. API designers in the field: Design practices and challenges for creating usable APIs. In: 2018 Ieee Symposium on Visual Languages and Human-Centric Computing (Vl/Hcc). IEEE, pp. 249–258.

Nguyen, Anh Tuan, Nguyen, Tien N., 2015. Graph-based statistical language model for code. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. IEEE.

Pautasso, Cesare, Wilde, Erik, 2009. Why is the web loosely coupled? A multifaceted metric for service design. In: Proceedings of the 18th International Conference on World Wide Web.

Pedro, San, De, Javier, Carmona, Josep, Cortadella, Jordi, 2016. Log-based simplification of process models. In: International Conference on Business Process Management. Springer, Cham.

Pržulj, Natasa, Corneil, Derek G., Jurisica, Igor, 2004. Modeling interactome: scale-free or geometric? Bioinformatics 20 (18), 3508–3515.

Sarajlić, Anida, Malod-Dognin, Noël, Yaveroğlu, Ömer Nebil, Pržulj, Nataša, 2016. Graphlet-based characterization of directed networks. Sci. Rep. 6 (1), 1–14.

Sohan, S.M., Anslow, Craig, Maurer, Frank, 2015. A case study of web API evolution. In: 2015 IEEE World Congress on Services. IEEE, pp. 245–252.

Suter, Philippe, Wittern, Erik, 2015. Inferring web API descriptions from usage data. In: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE.

Tanasa, Doru, Trousse, Brigitte, 2004. Advanced data preprocessing for intersites web usage mining. IEEE Intell. Syst. 19.

Van Der Aalst, Wil, 2012. Service mining: Using process mining to discover, check, and improve service behavior. IEEE Trans. Serv. Comput. 6 (4), 525–535.

Van Der Aalst, Wil MP, 2013. Challenges in service mining: record, check, discover. In: International Conference on Web Engineering. Springer, Berlin, Heidelberg, pp. 1–4.

Van Der Aalst, Wil, 2016. Data science in action. In: Process Mining. Springer, Berlin, Heidelberg.

Wang, Shaohua, Keivanloo, Iman, Zou, Ying, 2014. How do developers react to restful api evolution? In: International Conference on Service-Oriented Computing. Springer, Berlin, Heidelberg.

Wang, Jue, et al., 2013. Mining succinct and high-coverage API usage patterns from source code. In: 2013 10th Working Conference on Mining Software Repositories. MSR, IEEE.

Zhong, Hao, et al., 2009. MAPO: Mining and recommending API usage patterns. In: European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg.

Zibran, Minhaz F., Eishita, Farjana Z., Roy, Chanchal K., 2011. Useful, but usable? Factors affecting the usability of APIs. In: 2011 18th Working Conference on Reverse Engineering. IEEE.

Rediana Koçi received her M.Sc. in Information Systems from the University of Tirana in 2016. After few years working in industry, she started her Ph.D. studies (Doctoral Program in Computing) at the Universitat Politècnica de Catalunya (UPC-BarcelonaTech). Her research interest includes web API evolution, log analysis, process mining, and pattern mining.

Xavier Franch is a professor in Software Engineering at the Universitat Politècnica de Catalunya (UPC-BarcelonaTech). He received his Ph.D. degree in Informatics from UPC in 1996. His research interest embraces many fields in software engineering, including requirements engineering, empirical software engineering, open source software, and agile software development. He is a member of the IST, REJ, and Computing editorial boards. He served as a PC chair at CAiSE'22, RE'16, ICSOC'14, CAiSE'12, and REFSQ'11, among others, and as GC for RCIS'22, PROFES'19 and RE'08. More information at https://www.essi.upc.edu/~franch.

Petar Jovanovic obtained his Ph.D. from Universitat Politècnica de Catalunya (UPC) and Université libre de Bruxelles (ULB) in 2016. He is a tenure-track lecturer and a member of the INSSIDE research group at UPC. His research areas are Business Intelligence and Big Data, with special focus on big data management and data flow processing and optimization. He has published articles in peer reviewed international journals (such as TKDE, Information Systems, Fundamenta Informaticae), and conferences (such as ICDE, SIGMOD, EDBT, ADBIS, ER). He currently serves as committee member of several international conferences and journals. He has also participated in reserach projects with organizations such as WHO, HP Labs Palo Alto and in several H2020/Horizon Europe projects. He coordinates and teaches various B.Sc. and M.Sc. courses on databases and big data management.

Alberto Abelló is an Associate professor Ph.D. in Informatics, UPC. Local coordinator of the Erasmus Mundus Ph.D. program IT4BI-DC. Active researcher with more than 100 peer-reviewed publications and H-factor of 29, his interests include Data Warehousing and OLAP, Ontologies, NOSQL systems and BigData management. He has served as Program Chair of DOLAP and MEDI, being member also of the PC of other database conferences like DaWaK, CIKM, VLDB, etc.