Comparative study of xAPI validation tools

Thomas Rabelo[†], Manuel Lama[*], Juan C. Vidal[*], Ricardo Amorim[†‡]
[*] Centro de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain
[†] Faculdade de Ciências Aplicadas e Sociais de Petrolina (FACAPE)
[‡] Universidade do Estado da Bahia (UNEB), Brasil

***Abstract*—Learning Analytics (LA) is currently the most effective way of achieving better information and in-depth insights of the learning processes. Specifications like Experience API (xAPI) have been defined as part of LA initiatives to add interoperability among LA-aware applications. Tools that validate the conformance of data to these specifications are key components to assure the interoperability among applications. In this paper, a comparative evaluation of relevant validation tools of the xAPI specification is presented. This comparison focus specially on the structural and semantic features of the xAPI specification, revealing that most of the currently available tools do not support the semantic constraints of the specification.**

***Index Terms*—Big data architecture, Learning analytics, Experience API**

I. Introduction

Learning Analytics (LA) is understood as the measurement, collection and analysis of data on students and their context, and aims for understanding and optimizing the teaching and virtual environment in which it occurs [1]. To facilitate the development of LA-aware applications, some software architectures have been proposed [2], [3], [4], [5], [6], [7] which typically contain three main components: (i) a set of sensor API, which capture the data generated in the virtual learning environment (VLE) during a course; (ii) a set of analytic services, which process the information gathered by the sensors to obtain new insights about the student behavior, student performance prediction, and so on; and (iii) a graphical interface (or dashboard), which allows users to visualize and understand the results of analytic services in order to make decisions for improving the course.

In this scenario, it is necessary to standardize the representation of the data captured by the sensor API to facilitate the interoperability between LA-aware applications: if these data are in the same representation, analytic services only need to understand this representation, independently of the specific features of the VLE in which the courses are delivered. To deal with this issue, the xAPI specification [8] has emerged as the de facto standard to represent streams of data coming from virtual learning environments. In this specification, data are generated by an agent or group of agents (typically a student) that undertake an action (like create an input blog) in the context of a course. However, in order to ensure the interoperability between applications, it is needed to validate that data generated by a xAPI-based sensor API are compliant with the xAPI specification.

There are some proposals to validate xAPI data[9], [10], [11]. These proposals have two main drawbacks: (i) they have not been developed for processing a high number of statements, and, therefore, they could not been applied in learning analytic architectures that support a large number of students (like MOOC); and (ii) they validate the structural and syntactic features of the xAPI specification, but do not take into account its semantic description, which is described in natural language in the specification document. This paper focus on this second drawback.

In this paper, a comparative evaluation of the most relevant xAPI validators currently available is presented. This comparison focus not only on the structural and syntactical properties of the xAPI specification, but also on the semantic information provided by the standard. For enabling this comparison, a total of 116 fixture

II. xAPI Validation tools

Data quality is a key issue in Learning Analytics (LA), and xAPI plays an important role in this process when collecting and storing events in the LRS. For this reason, many xAPI validators have been developed recently to ensure this quality and the compliance of xAPI statements. In this paper we will analyse the behaviour of the most relevant validators. Other validators, such as *Statement Factory*[1], *xAPI validator*[2], or *TinCanValidator*[3], were also considered for this study but were discarded mainly because of issues in execution time or with JSON serialization. As it is depicted in Table I, four tools were finally selected:

- *Yet Analytics* provides a dynamic and interactive analytics platform based on xAPI, and a library[4], for Clojure and JavaScript, which schema is compatible with the latest version of xAPI (presently 1.0.3). The source code is available on GitHub under Eclipse license.
- *Learning Locker* is an open source Learning Record Store, developed by HT2 Labs, a key implementer of xAPI-enabled software. It provides two separate validation tools: a PHP server-side library (for xAPI version 1.0.0) and a newer release in Javascript[5] (for xAPI version

[1]Available at https://github.com/LearningLocker/StatementFactory
[2]Avaliable at https://github.com/php-xapi/xapi-validator
[3]Available at https://github.com/RusticiSoftware/TinCanValidator
[4]Available at https://github.com/yetanalytics/xapi-schema
[5]Available at https://github.com/LearningLocker/StatementFactory/

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

Table I
XAPI VALIDATION TOOLS

Tool	Type	xAPI Version	License	Source Language
Yet Analytics	Library	1.0.3	Eclipse	Closure
Learning Locker JavaScript	Library	1.0.3	GNU	Javascript
ADLNET	Library	1.0.0	Apache 2.0	Javascript
SmartLAK	Service	1.0.3	N/A	Java

A "+" indicates that the tool supports the feature, "+/-" that it is partially supported, while "-" that the feature is not supported.

1.0.3). The source code of both validators is available on GitHub under GNU license.

- *ADLNET*[6] is a Javascript library, available on GitLab under MIT license. This library, created by ADL (one of the main contributors in the development of xAPI) supports the version 1.0.0 of xAPI and can be used both on the client or server-side.
- *SmartLAK*[7] is an ontology-based approach for xAPI statements validation. Contrary to the other approaches, it provides a set of REST-based services for validation instead of a library. SmartLAK supports the latest version of xAPI (1.0.3).

Table II describes other important features from the perspective of usability. The lack of documentation is an important issue, with the exception of *SmartLAK* that provides a complete description of the services and a step-by-step tour in order to guide users on its Client Web Interface. The other validators only help the developer with a README file. In the case of *Learning Locker* and *ADLNET*, this file only provides a usage example.

Allowing the validation of multiple statements or an entire dataset is another key feature that these tools should support, since different datasets may have different data quality. This kind of analysis is essential to guarantee the correct alignment of the datasets and thus the performance of LA techniques. All the validators but *Learning Locker* support this feature.

As their main objective, all the tools support error tracking. However, the identification of errors should be complemented by a precise error reporting and internationalization. This improves the quality of the information returned to users when an error is detected during the validation process. ADLNET and SmartLAK generate error reports to explain what is the cause that motivated the identification of the error, in addition to the global statistics of the validation process.

III. METHODOLOGY

xAPI specification provides a text-based description of the data model and verbalizes some constraints over the elements of this model. However, the specification never gets to formalize the model nor clearly identify its constraints. This is an important drawback since informal models trend to ambiguity, and thus are error prone. Taking this into account, in this paper we use the xAPI ontology proposed in [12] to have a more solid foundation during the experimentation. This is a heavy ontology which formalizes the xAPI specification in OWL [13] (a W3C recommendation for describing ontologies) and, in addition, explicitly identifies the axioms that constrain the model.

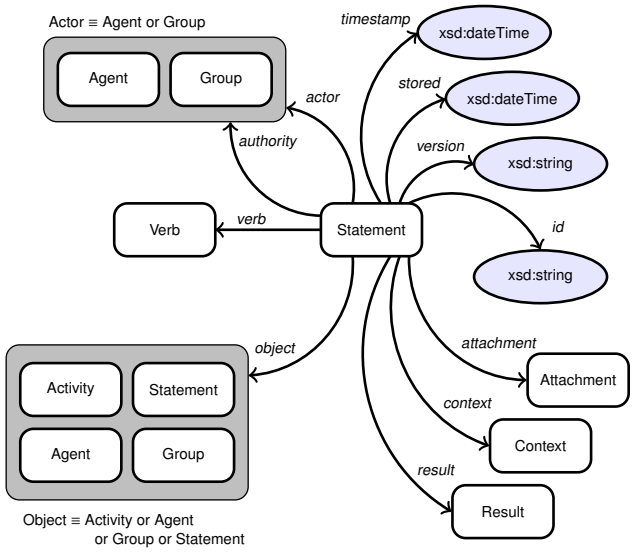

Figure 1. Semantic network that represents an xAPI *statement*

As example, Figure 1 represents, though a semantic network, the classes and relations that describe an xAPI statements such as "John wrote an essay about Ancient Egypt". In this case, "John" is identified as the actor, "wrote" as the verb, and "an essay about Ancient Egypt" as the object of the statement. However, this semantic network is a simplified view of the model. If we go further, many axioms are needed to guarantee that, e.g., a statement has only one actor. This type of axioms is represented in Table III and are used to restrict the classes and their properties. For example, the first row indicates that there is a property, called *actor*, between a statement and an actor class (*Agent* or *Group*), and that a statement can have only one instance of this property.

More complex axioms, such as "a sub-statement cannot have a version attribute" (it is assumed that sub-statements have the same version as the main statement), cannot be expressed in OWL. However, the ontology includes additional axioms, in SWRL [14], to represent these constraints. For instance, Table IV shows some of the axioms that verify this complex semantics of statements. As we can see, the former

[6]Available at https://github.com/adlnet/xAPI-Validator-JS
[7]Available at https://tec.citius.usc.es/smartlak/xapi/validator/#/

Table II
SOME FEATURES OF THE ANALYZED XAPI VALIDATION TOOLS

Tool	Documentation	Demo	Multiple statement	Internationalization	Error tracking	Error report
Yet Analytics	+/-	+	+	+	+	-
Learning Locker JavaScript	-	-	-	-	+	-
ADLNET	-	+	+	-	+	+
SmartLAK	+	+	+	-	+	+

A "+" indicates that the tool supports the feature, "+/-" that it is partially supported, while "-" that the feature is not supported.

Table III
EXAMPLE OF THREE OWL AXIOMS THAT CONSTRAIN STATEMENTS CLASS

	Class			
#	Domain	Range	Property	Axiom
S1	Statement	Actor	actor	*exactly 1*
S2		Verb	verb	*exactly 1*
S3		Object	object	*exactly 1*

Table IV
EXAMPLE OF TWO SWRL RULES THAT CONSTRAINT STATEMENTS

#	Axiom

S11	Sub-statements must have a *id* but no *stored*, *version*, or *authority* properties:
	`Statement(?a), Statement(?b), object(?a, ?b) ->`
	`(authority exactly 0 foaf:Agent)(?b),`
	`(stored exactly 0 xsd:dateTime)(?b),`
	`(version exactly 0 xsd:string)(?b),`
	`(id exactly 1 xsd:string)(?b)`

S12	Sub-statements cannot be nested:
	`Statement(?a), Statement(?b), object(?a, ?b),`
	`object(?b, ?c) -> (not (Statement))(?c)`

Table V
TEST CASES FOR THE VERIFICATION OF THE AXIOM S11 (TURTLE SYNTAX)

#		Test case

1	✓	`ex:s1 rdf:type xapi:Statement .`
		`ex:s2 rdf:type xapi:Statement .`
		`ex:s1 xapi:object ex:s2 .`
		`ex:s2 xapi:uuid "222"^^xsd:string .`

2	✗	`ex:s1 rdf:type xapi:Statement .`
		`ex:s2 rdf:type xapi:Statement .`
		`ex:s1 xapi:object ex:s2 .`
		`ex:s2 xapi:uuid "222"^^xsd:string .`
		`ex:a1 rdf:type foaf:Agent .`
		`ex:s2 xapi:authority ex:a1 .`

3	✗	`ex:s1 rdf:type xapi:Statement .`
		`ex:s2 rdf:type xapi:Statement .`
		`ex:s1 xapi:object ex:s2 .`
		`ex:s2 xapi:uuid "222"^^xsd:string .`
		`ex:s2 xapi:stored "2001-10-26T21:32:52" .`

4	✗	`ex:s1 rdf:type xapi:Statement .`
		`ex:s2 rdf:type xapi:Statement .`
		`ex:s1 xapi:object ex:s2 .`
		`ex:s2 xapi:uuid "222"^^xsd:string .`
		`ex:s2 xapi:version "v2.2"^^xsd:string .`

axiom is represented in the first row.

Most of the axioms of the ontology were used to define the test fixtures for the comparison. However, some axioms were discarded because they are already included in structural validation of JSON schema. For each of the selected axioms we created at least a correct and an incorrect test case to ensure that the axiom does not trigger false negatives or false positives, respectively. For example, Table V shows the test cases used to verify the previously mentioned axiom S11. In this case, the first test verifies that the sub-statement can be defined correctly, while the other three check that a sub-statement cannot have an authority, stored, or version property, respectively. A total of 116 test fixtures were created[8]. These fixtures were grouped in 7 categories based on the different parts of the xAPI model: *statements* (13 fixtures), *agents* (28 fixtures), *verbs* (2 fixtures), *activities* (45 fixtures), *results* (6 fixtures), *contexts* (12 fixtures), and *attachments* (10 fixtures).

IV. RESULTS

Table VI summarizes the results obtained by the selected validators for each xAPI model in terms of fixtures correctly managed. For instance, the first column of the upper part of the table contains the percent coverage for the statements model. As we can see, the first three validators are able to process correctly 84.61% of the fixtures. Only *SmartLAK* reaches full coverage. The detail of this assessment is represented in Table VII where a ✓ in the case type column represents that a *correct* while a ✗ an incorrect test case. If both markers are in the column, then the evaluation of the fixture applies to both correct and incorrect cases. On the other hand, a "+" indicates that the tool supports the feature, while "-" that the feature is not supported. From these results, we can infer that only *SmartLAK* supports the validation of sub-statements. In fact, all the tools but *SmartLAK* fail to validate a correctly formatted sub-statement (rows 4 and 9). The fact that these tools are able to validate incorrect sub-statements (rows 5-8 and 10) should not mislead us because they evaluate as an incorrect all the sub-statement.

Most of the xAPI models are correctly handled by the four validators:

- *Agents model*. *ADLNET* fails in 2 of the 28 fixtures: when the agent ID is an *openid* and when the group ID is an *account*.
- *Verbs model*. All validators verify the fixtures.
- *Results model*. *Learning Locker* fails in 2 of the 6 fixtures:

[8]The dataset of test fixtures is available at https://gitlab.citius.usc.es/smartlak/smartlak-xapiontology-fixtures/tree/master

Table VI
VALIDATORS COVERAGE OF TEST FIXTURES

Tool	Statements	Agents	Verbs	Activities
Yet Analytics	84.61%	100%	100%	77.77%
Learning Locker	84.61%	100%	100%	51.11%
ADLNET	84,61%	89.28%	100%	64.44%
SmartLAK	100%	100%	100%	100%

Tool	Results	Contexts	Attachments
Yet Analytics	100%	25.00%	100%
Learning Locker	66.66%	91.66%	100%
ADLNET	100%	58.33%	100%
SmartLAK	100%	100%	100%

Table VII
FIXTURES TESTED FOR THE STATEMENTS MODEL

A-ID	Objective	Case type		YA	LL	ADL	SL
S1	property:actor	✓	✗	+	+	+	+
S2	property:verb	✓	✗	+	+	+	+
S3	property:object	✓	✗	+	+	+	+
S11	substatement	✓		-	-	-	+
S11	substatement		✗	+	+	+	+
S11	substatement:stored		✗	+	+	+	+
S11	substatement:version		✗	+	+	+	+
S11	substatement:authority		✗	+	+	+	+
S12	substatement:nested	✓		-	-	-	+
S12	substatement:nested		✗	+	+	+	+

YA stands for *Yet Analytics*, *LL* for *Learning Locker*, *ADL* for *ADLNET*, and *SL* for *SmartLAK*.

Table VIII
FIXTURES NOT VERIFIED FOR THE CONTEXTS MODEL

A-ID	Objective	Case type	YA	LL	ADL	SL
C8a	property:revision	✗	-	+	-	+
C8b	property:revision	✗	+	+	-	+
C8c	property:revision	✗	+	+	-	+
C8d	property:revision	✗	-	+	-	+
C9a	property:platform	✗	-	+	-	+
C9b	property:platform	✗	+	+	-	+
C9c	property:platform	✗	+	+	-	+
C9d	property:platform	✗	-	+	-	+
C10	property:actor	✓	-	-	-	+

YA stands for *Yet Analytics*, *LL* for *Learning Locker*, *ADL* for *ADLNET*, and *SL* for *SmartLAK*.

it allows incorrect values for *raw* and *max* properties.

- *Attachments model*. All validators verify the fixtures.

On the contrary, validators have more problems dealing with the semantics of contexts and activities. In the case of contexts, all but *SmartLAK* fail to verify some of the fixtures. These fixtures are represented in Table VIII. As we can see, *Yet Analytics*, *Learning Locker*, and *ADLNET* fail at some point to validate incorrect values of the *revision* and *platform* (rows 1-8). Specifically, when the object of the statement is not an activity. The last row (axiom C10) represents the case in which the statement and the contextual information are the same, and is only supported by *SmartLAK*.

Finally, Table IX shows a small part of the fixtures that were not verified by at least one of the validators. Most of these fixtures are related to interaction activities defined in the xAPI specification. For example, the rows represented in this table only check a minimum cardinality for the *choices*, *scale*, *target*, *source*, and *steps* properties when they are associated to a choice, sequencing, likert, matching, or performance interaction activity, respectively. Most of the 26 fixtures that failed the verification and that are not represented in the former table are also related to the interaction activities. As example, we can mention the followings:

Table IX
FIXTURES NOT VERIFIED FOR THE ACTIVITIES MODEL

A-ID	Objective	Case type		YA	LL	ADL	SL
A6	property:choices	✓	✗	+	-	+	+
A7	property:choices	✓	✗	+	-	+	+
A8	property:scale	✓	✗	+	-	+	+
A9	property:source	✓	✗	+	-	+	+
A10	property:target	✓	✗	+	-	+	+
A11	property:steps	✓	✗	+	-	+	+

YA stands for *Yet Analytics*, *LL* for *Learning Locker*, *ADL* for *ADLNET*, and *SL* for *SmartLAK*.

- A performance value with no *string answer* must have a *number answer*.
- A performance value with no *numeric answer* must have a *string answer*.
- The response of a *choice* activity must match the *id* of one of the interaction components.
- The response of a *matching* activity must contain one of the source *id* and one of the target *id*.

V. CONCLUSIONS

In this paper, a detailed study of the four most representative validation tools for xAPI is presented. The tools considered were *Yet Analytics*, *Learning Locker*, *ADLNET*, and *SmartLAK*. The results have clearly proved that *SmartLAK* is the most suitable tool to perform the validation of xAPI statements. In fact, it is the only tool that supports the fixtures designed to test the validators compliance with xAPI. The other tools have clearly failed to verify 3 of the 7 parts of the xAPI data model: statements, contexts, and activities. In addition, SmartLAK provides a GUI from which users can directly check their JSON-formatted xAPI statements or directly verify a complete dataset stored in a LRS.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Economy and Competitiveness under the projects TIN2014-56633-C3-1-R and TIN2015-73566-JIN and by the European Regional Development Fund (ERDF/FEDER) under the project CN2012/151 of the Galician Ministry of Education. In addition, it was supported by Instituto Nacional de Ciencia e Tecnologia para Engenharia de Software (INES) with financial support of CNPq under the project 465614/2014-0.

REFERENCES

[1] G. Siemens and D. Gasevic, "Guest editorial-learning and knowledge analytics," *Educational Technology & Society*, vol. 15, no. 3, pp. 1–2, 2012.

[2] B. Vázquez-Barreiros, M. Lama, M. Mucientes, and J. C. Vidal, "Softlearn: A process mining platform for the discovery of learning paths," in *IEEE 14th International Conference on Advanced Learning Technologies (ICALT 2014)*. IEEE Computer Society, 2014, pp. 373–375.

[3] A. Rayón, M. Guenaga, and A. N. nnez, "Ensuring the integrity and interoperability of educational usage and social data through Caliper framework to support competency-assessment," in *44th Annual Frontiers in Education Conference (FIE 2014)*. IEEE Press, 2014, pp. 2275–2783.

[4] W. Greller and H. Drachsler, "Translating learning into numbers: A generic framework for learning analytics," *Educational Technology & Society*, vol. 15, no. 3, pp. 42–57, 2012.

[5] IMS GLOBAL Learning Consortium, "Learning measurement for analytics whitepaper," http://imsglobal.org/IMSLearningAnalyticsWP.pdf, 2013.

[6] Z. A. Pardos, A. Whyte, and K. Kao, "moocRP: Enabling Open Learning Analytics with an Open Source Platform for Data Distribution, Analysis, and Visualization," *Technology, Knowledge and Learning*, vol. 21, no. 1, pp. 75–98, 2016.

[7] N. Sclater, A. Berg, and M. Webb, "Developing an open architecture for learning analytics," *Proceedings of the EUNIS 2015 Congress*, 2015.

[8] Advanced Distributed Learning (ADL), "Experience API. Version 1.0.1," http://www.adlnet.gov/wp-content/uploads/2013/10/xAPI_v1.0.1-2013-10-01.pdf, October 2013.

[9] Rustic Software, "JavaScript Utilities to do validation of Tin Can structures based off of TinCanSchema." [Online]. Available: https://github.com/RusticiSoftware/TinCanValidator

[10] Yet Analytics, "Clojure(script) schema for the Experience API." [Online]. Available: https://github.com/yetanalytics/xapi-schema

[11] HT2 Labs, "Learning Lockers classes for xAPI," 2015. [Online]. Available: https://github.com/LearningLocker/StatementFactory/

[12] J. C. Vidal, T. Rabelo, and M. Lama, "Semantic description of the Experience API specification," in *IEEE 15th International Conference on Advanced Learning Technologies (ICALT 2015)*. IEEE Computer Society, 2015.

[13] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," W3C, W3C Recommendation, Feb. 2004.

[14] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C W3C Member Submission, May 2004.