Versioning Temporal Characteristics of JSON-based Big Data via the τJSchema Framework

Versioning Temporal Characteristics of JSON-based Big Data via the τJSchema Framework

Safa Brahmia, Zouhaier Brahmia*

Department of Computer Science, Faculty of Economics and Management, University of Sfax,
Road of the Aerodrome, Km 4.5, P.O.Box 1088, 3018 Sfax, Tunisia
E-mail: safa.brahmia@gmail.com
E-mail: zouhaier.brahmia@fsegs.rnu.tn
*Corresponding author

Fabio Grandi

DISI - Department of Computer Science and Engineering, University of Bologna
Viale Risorgimento, 2, I-40136 Bologna BO, Italy
E-mail: fabio.grandi@unibo.it

Rafik Bouaziz

Department of Computer Science, Faculty of Economics and Management, University of Sfax,
Road of the Aerodrome, Km 4.5, P.O.Box 1088, 3018 Sfax, Tunisia
E-mail: rafik.bouaziz@usf.tn

Abstract: In previous work, we have proposed the use of a framework, named τJSchema (temporal JSON Schema), for the definition and validation of temporal JSON documents that conform to a temporal JSON schema. A τJSchema schema is composed of a conventional (i.e., non-temporal) JSON schema, annotated with a set of temporal logical and temporal physical characteristics. Subsequently, we have extended τJSchema to support versioning of conventional JSON schemas. In this work, we complete the picture by extending our framework to also support versioning of temporal characteristics. In fact, we propose a suitable versioning technique and provide a complete set of low-level change operations for the maintenance of these characteristics; for each operation, we define its arguments and its operational semantics. With this extension, τJSchema provides a full support of temporal versioning of JSON-based Big Data at both instance and schema levels.

Keywords: Big Data, NoSQL, JSON, JSON Schema, τJSchema, Conventional JSON schema, Temporal JSON schema, Temporal logical characteristic, Temporal physical characteristic, Schema change, Schema versioning

Reference to this paper should be made as follows: Brahmia, S., Brahmia, Z., Grandi, F. and Bouaziz, R. (xxxx) 'Versioning Temporal Characteristics of JSON-based Big Data via the τJSchema Framework', *Int. J. Cloud Computing*, Vol. , No. , pp. .

Biographical notes: Safa Brahmia is currently a PhD student in Computer

Science at the Faculty of Economics and Management of the University of Sfax, Tunisia. She is working on temporal and multi-schema-version JSON-based NoSQL databases. She is a member of the Multimedia, InfoRmation systems, and Advanced Computing Laboratory (MIRACL) since 2015. She received her MSc degree in Computer Science, in December 2014, from the Faculty of Economics and Management of the University of Sfax.

Zouhaier Brahmia is currently an Associate Professor of Computer Science in the Department of Computer Science at the Faculty of Economics and Management of the University of Sfax, Tunisia. He is a member of the Multimedia, InfoRmation systems, and Advanced Computing Laboratory (MIRACL). His scientific interests include temporal databases, schema versioning, and temporal, evolution and versioning aspects in emerging databases (XML, and NoSQL), Big Data, and Semantic Web ontologies. He received an MSc degree in Computer Science, in July 2005, and a PhD in Computer Science, in December 2011, from the Faculty of Economics and Management of the University of Sfax.

Fabio Grandi has been an Associate Professor at the University of Bologna, Italy, since 1998, currently in the Department of Computer Science and Engineering. From 1989 to 2012 he has worked at the CSITE center of the Italian National Research Council (CNR) in Bologna in the field of neural networks and temporal databases, initially supported by a CNR fellowship. In 1993 and 1994 he was an Adjunct Professor at the Universities of Ferrara, Italy, and Bologna. He was appointed as Research Associate at the University of Bologna in 1994. His scientific interests include temporal, evolution and versioning aspects in data management, WWW and Semantic Web, knowledge representation, storage structures and access cost models. He received a Laurea degree cum Laude in Electronics Engineering and a PhD in Electronics Engineering and Computer Science from the University of Bologna.

Rafik Bouaziz is currently a Full Professor of Computer Science at the Faculty of Economics and Management of the University of Sfax, Tunisia. He was the president of the University of Sfax between 2014 and 2017, and the director of the "Economy, Management, and Computer Science" doctoral school, in the same university, between 2011 and 2014. In his PhD thesis (defended in 1988), he has dealt with temporal data management and historical record of data in Information Systems. The subject of his habilitation (obtained in 2007) was "A contribution for the management of data and schema versioning in advanced information systems". His current research interests are temporal databases, real-time databases, information systems engineering, ontologies, data warehousing and workflows.

This paper is a revised and expanded version of a paper entitled 'Managing temporal and versioning aspects of JSON-based big data via the τJSchema framework' presented at The International Conference on Big Data and Smart Digital Environment (ICBDSDE'2018), Casablanca, Morocco, 29–30 November 2018.

1 Introduction

Nowadays, Big Data (Chen and Zhang, 2014; IRMA, 2016; Khosla and Kaur, 2018) are being exchanged and stored in the Cloud to be used in various applications like Internet of Things, healthcare systems, social networks, big science projects and Smart Cities.

Moreover, both structures (or definitions) of Big Data and their instances are evolving over time and changing at a very high speed, to reflect changes in users' requirements or in the reference world of the database. Moreover, several modern applications, which exploit Big Data, require a complete history of all Big Data versions and all changes performed on both data instances and their schemas (Cuzzocrea, 2015), in order to allow (i) recovering past Big Data versions, (ii) tracking changes over time, and (iii) executing temporal queries (Snodgrass et al., 1995) on temporal Big Data. However, although the schema versioning technique —which consists in creating a new schema version each time a schema change is applied, while preserving old schema versions with their corresponding data (Brahmia et al., 2015)— has long been advocated to be the best solution to this issue, no technical support is currently available. In fact, state-of-the-art Big Data management systems, especially NoSQL database management systems[1] (Cattell, 2010; Tiwari, 2011; Pokorný, 2013; Sharma et al., 2014; Gudivada et al., 2014; Sharma et al., 2015; Corbellini et al., 2017; Davoudian et al., 2018), do not provide solutions for handling either temporal evolution and versioning aspects of Big Data. Therefore, the designers and developers of the aforementioned applications have to proceed in an ad hoc manner when they should deal with Big Data evolution while keeping track of all versions of Big Data and their schemas, or when they should allow time-slice queries to be evaluated, or when it is required to specify a comprehensive schema for time-varying Big Data.

In order to efficiently manage and query Big Data evolution over time, we think that we should have Big Data management systems with built-in temporal support. To this purpose, in our previous work (Brahmia et al., 2016), we have proposed the adoption of a framework named τJSchema (temporal JSON Schema) for a disciplined and systematic approach to the temporal management of JSON-based Big Data in NoSQL databases. The τJSchema infrastructure includes a data model and suite of tools that allow the NoSQL Database Administrator (NSDBA) to create and validate temporal JSON documents (which store time-varying Big Data) through the use of a temporal JSON schema (which defines the structure of these temporal Big Data and to which obey the temporal JSON documents). A τJSchema schema consists in a conventional (i.e., non-temporal) JSON schema (IETF, 2013a) annotated with a set of temporal logical and temporal physical characteristics. Changing the definition of the temporal characteristics, according to the evolution of the application requirements can make a big difference in the resulting temporal Big Data representation. It is worth mentioning that the temporal logical and physical characteristics are orthogonal and are independently maintained, while they are stored together in a single JSON document (IETF, 2014), named the temporal characteristics document and associated to the conventional JSON schema.

In its initial definition (Brahmia et al., 2016), τJSchema was proposed as an infrastructure for managing JSON documents with time-varying instances that are valid to a static schema; only instance versioning was supported at that stage. Nevertheless, since each one of the three components of a τJSchema schema —namely conventional JSON schema, temporal logical characteristics, and temporal physical characteristics— could also evolve over time to respond to new applications' requirements, in (Brahmia et al., 2017) we augmented the τJSchema capabilities with the support of versioning of

[1] http://www.nosql-database.org/

conventional JSON schemas. In this work, as in its preliminary version (Brahmia et al., 2018a), we complete the picture by extending our framework to also support versioning of temporal logical and temporal physical characteristics. Indeed, we propose a technique for temporal characteristics versioning, and provide a complete set of low-level change operations for the maintenance of these characteristics. For each one of the proposed operations, we define its arguments and its operational semantics. Thus, with this extension, τJSchema will fully support temporal versioning of JSON-based Big Data at both instance and schema levels, and consequently will provide bookkeeping of a fully-fledged history of Big Data changes.

The remainder of this paper is organized as follows. Section 2 briefly describes the τJSchema framework. Section 3 presents our approach for versioning of temporal (logical and physical) characteristics. Section 4 introduces the schema change operations that we propose for the maintenance of temporal logical and physical characteristics. Section 5 illustrates our approach through an application example. Section 6 discusses related work. The last section summarizes the paper and gives some remarks about our future work.

2 The τJSchema Framework

In this section, we provide a detailed description of our τJSchema framework (Brahmia et al., 2016) for handling temporal JSON documents. Its definition was inspired by the well-known τXSchema framework proposed for XML documents in (Currim et al., 2004; Snodgrass et al., 2008) and further developed in (Brahmia et al., 2014; Brahmia et al., 2018b), with which it shares the same organization.

The τJSchema framework allows a NSDBA to create a temporal JSON schema for temporal JSON data instances from a conventional JSON schema, temporal logical characteristics, and temporal physical characteristics. The τJSchema organization is based on the following principles: (i) separation between the conventional schema and the temporal schema, and between the conventional instances and the temporal instances; (ii) use of logical and physical characteristics to specify temporal logical and temporal physical aspects, respectively, at schema level.

Figure 1 illustrates the architecture of τJSchema. Notice that only the components that are presented in the figure as rectangular boxes with one continuous line border (i.e., boxes 1, 2, 3, and 4) are specific to an individual time-varying JSON document and need to be supplied by a NSDBA. The framework is based on the JSON Schema language (IETF, 2013a).

The NSDBA starts by creating the conventional JSON schema (box 1), which is a traditional JSON Schema document that models a given real world entity, without any temporal aspect. To each conventional JSON schema corresponds a set of conventional (i.e., non-temporal) JSON documents or JSON Schema instances (box 2). Any change to the conventional JSON schema is propagated to its corresponding instances.

After that, the NSDBA augments the conventional schema with temporal logical and temporal physical characteristics, which allow him/her to express, in an explicit way, all requirements dealing with the representation and the management of temporal aspects associated to the components of the conventional schema, as described below.

Temporal logical characteristics (Currim et al., 2004) allow the NSDBA to specify (i) whether a conventional schema component (e.g., property, object) varies over valid time

and/or transaction time, (ii) whether its lifetime must be described as a continuous state or a single event, (iii) whether the component itself may appear at certain times (and not at others), and (iv) whether its content changes. If no logical characteristics are provided, the default logical characteristic is that anything can change. However, once the conventional schema is annotated, components that are not described as time-varying are static and, thus, they must have the same value across every conventional JSON document instance (box 2).

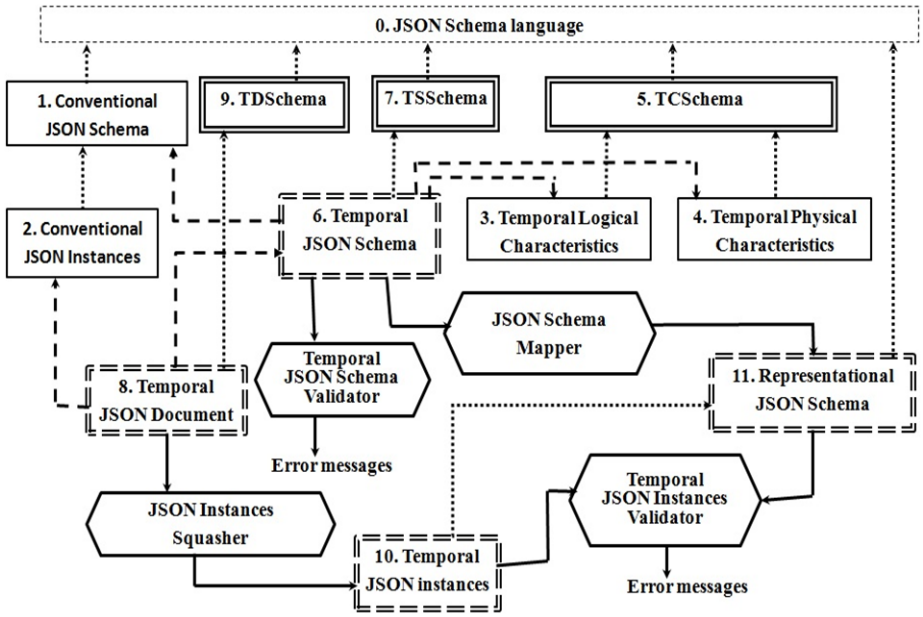

Figure 1 The architecture of τJSchema (Brahmia et al., 2016).

Temporal physical characteristics (Currim et al., 2004) specify the timestamp representation options chosen by the NSDBA, such as where the timestamps are placed and which kind of representation is adopted for their implementation.. The location of timestamps is largely independent of which components vary over time. Timestamps can be located either on time-varying components (as specified by the logical characteristics) or somewhere above such components in the data structure hierarchical organization. Two JSON documents with the same logical characteristics will look very different if we change locations of their physical timestamps. Changing some aspects of even one timestamp may give rise to very different representations. τJSchema supplies a default set of physical characteristics, which is to timestamp the root property with valid and transaction times. However, explicitly defining them can lead to more compact representations (Currim et al., 2004).

Although the two sets of temporal characteristics are orthogonal and can evolve independently, they are stored together in a single JSON document associated to the conventional schema which is a standard JSON document named the temporal characteristics document. The schema for the logical and physical characteristics is given by TCSchema (box 5) which is JSON Schema document (IETF, 2013a).

Finally, the NSDBA finishes by annotating the conventional schema and asks the system to save his/her work. Consequently, the system creates the temporal JSON schema (box 6) providing the linking information between the conventional schema and its corresponding logical and physical characteristics. The temporal schema is a standard JSON document, which ties the conventional schema, the logical characteristics, and the physical characteristics together. In the τJSchema framework, the temporal JSON schema is the logical equivalent of the conventional JSON schema in a non-temporal environment. This document associates a series of conventional schema definitions with temporal characteristics, along with the time span during which the association was in effect. The schema for the temporal JSON schema document is given by TSSchema (box 7) which is JSON Schema document.

After creating the temporal schema, the system creates a temporal JSON document (box 8) in order to link each conventional JSON document (box 2), which is valid to a conventional JSON schema (box 1), to its corresponding temporal JSON schema (box 6), and more precisely to its corresponding logical and physical characteristics (which are referenced by the temporal JSON schema). A temporal document is a standard JSON document that maintains the evolution of a non-temporal JSON document over time, by keeping track of all the versions (or temporal slices) of the document with their corresponding timestamps and by specifying the temporal schema associated to these versions. This document associates a series of conventional JSON documents with logical and physical characteristics, along with the time span during which the association was in effect. Therefore, the temporal JSON document facilitates the support of temporal queries involving past JSON document versions or dealing with changes between JSON document versions. The schema for the temporal document is the JSON Schema document TDSchema (box 9).

Notice that, whereas TCSchema (box 5), TSSchema (box 7), and TDSchema (box 9) have been developed in this work, JSON Schema (box 0) corresponds to the existing language endorsed by the Internet Engineering Task Force (IETF, 2013a) for specifying the structure of JSON documents (IETF, 2014).

The temporal JSON schema (box 6) is processed by the temporal JSON schema validator tool in order to ensure that the logical and physical characteristics are (i) valid with respect to TCSchema, and (ii) consistent with the conventional schema. The temporal JSON schema validator tool reports whether the temporal JSON schema document is valid or invalid.

Once all the characteristics are found to be consistent, the JSON schema mapper tool generates the representational JSON schema (box 11) from the temporal JSON schema (i.e., from the conventional JSON schema plus the logical and physical characteristics); it is the result of transforming the conventional schema according to the requirements expressed through the different temporal characteristics. The representational JSON schema becomes the schema for temporal JSON data instances (box 10). These temporal instances could be obtained in four ways:

(i) automatically from the temporal JSON document (box 8) (i.e., from non-temporal JSON instances (box 2) and the temporal JSON schema (box 6)), using the JSON instances squasher tool;
(ii) automatically from instances stored in a JSON-based NoSQL database, that is as the result of a “temporal query” or a “temporal view”;
(iii) automatically from a third-party tool;
(iv) manually: temporal JSON instances are directly added by the NSDBA to the

τJSchema repository.

Moreover, temporal JSON instances are validated against the representational JSON schema through the temporal JSON instances validator tool, which reports whether the temporal JSON instances (box 10) are valid or invalid.

The four mentioned tools (i.e., Temporal JSON Schema Validator, Temporal JSON Instances Validator, JSON Schema Mapper, and JSON Instances Squasher) are currently under development. For example, the temporal JSON instances validator tool is being implemented as a temporal extension of an existing conventional JSON instances validator (IETF, 2013b).

3 Versioning of Temporal Logical and Physical Characteristics

In this section, we describe how τJSchema logical and physical characteristics are versioned in our approach.

The first step of a schema versioning process is the creation of the first temporal JSON schema version: the NSDBA creates a conventional JSON Schema document (i.e., a classical JSON Schema file) annotated with some logical and physical characteristics in an independent document (which is also stored as a JSON file). Consequently, the system generates the temporal JSON schema (also stored as a JSON file) that ties together the conventional schema and the temporal characteristics. In further steps of the versioning process, when necessary, the NSDBA can independently change the conventional schema, the temporal logical characteristics or the temporal physical characteristics.

Changing the conventional schema leads to a new version of it. Similarly, changing temporal logical and/or physical characteristics leads to a new version of the whole temporal characteristics document. Therefore, the temporal JSON schema is automatically updated after each change to the conventional JSON schema or to the temporal characteristics document, in order to take into account the new version of the corresponding changed component. In this paper, we do not deal with changes to the conventional schema.

Notice that change operations performed by the NSDBA are in general high-level, since they are usually conceived having in mind high-level real-world object properties. However, in this work, we focus on the definition of low-level primitive change operations, since high-level change operation can be expressed as the composition of low-level change operations that can be applied in sequence. To this end, we will propose a complete set of low-level operations by means of which the NSDBA will be able to perform any simple/complex change to the temporal characteristics document (based on the structure of this latter).

4 Operations for Changing Temporal Logical and Physical Characteristics

In this section, we define low-level operations for changing temporal logical and physical characteristics in τJSchema. For each one of these operations, we provide a description of its arguments and its operational semantics. The definition of all these operations is based on (i) the schema (or the structure) of the temporal characteristics document (TCD) that we have constructed, as explained in the first subsection, titled “The Schema of Temporal

Characteristics Documents", and (ii) some common design choices that we have introduced in the second subsection, titled "Design Choices".

4.1 The Schema of Temporal Characteristics Documents

In the architecture of our τJSchema framework (Brahmia et al., 2016), the schema for the temporal logical and physical characteristics is given by TCSchema (box 5) which is a JSON Schema (IETF, 2013a) file that describes the structure of any temporal characteristics document. This schema has been only mentioned in our previous work (Brahmia et al., 2016) without being provided. The full listing of its JSON Schema code can be found in Figure A1 of Appendix 1.

The analysis of TCShema allows us to determine the list of components of any TCD (e.g., logical, logicalItems, timeDimension, validTime, transactionTime, physical, stamps, stampKind, stampBounds). Based on this list, we will propose all possible change operations that could be executed on each component, by creating, modifying or dropping it.

4.2 Design choices

The definition of the primitives will obey the following design choices:

- All operations must have a valid temporal characteristics document (TCD) as input and must produce a valid TCD as output.
- All operations need to work on a JSON file storing the TCD, whose name must be supplied as first argument.
- For all operations, arguments which are used to identify the object on which the operation works are in the first place of the argument list.
- Components which are just containers for other components (e.g., logical, physical) can be managed by the operations concerning the components, without specific operations acting on them (i.e., the container is created when the first sub-component is created and is deleted when the last sub-component is deleted).
- Operations adding objects with possibly optional properties have the values for all the properties as arguments; empty places in the argument list stand for unspecified optional properties.
- We use Add…/Change… operations for all components (objects or properties) which have multiple occurrences (e.g., "logicalItem", "stamp"); a single Set… operation is used for adding/changing components with occurrences <=1 (e.g., "validTime", "stampKind").

By applying the design choices presented above and taking into account the different components of the TCSchema file, we have defined a set of forty (40) low-level operations. We have organized this set into four subsets: (i) operations acting on the entire TCD, (ii) operations that are common to the logical and to the physical characteristics, (iii) operations that are specific to the logical characteristics, and (iv) operations that are specific to the physical characteristics.

For each change operation, we describe its arguments and its operational semantics. Obviously, each operation change has an effect on the TCD. Due to space limitations, we

do not present in this paper the effects of all change operations but only of some selected operations.

4.3 Operations Acting on the Whole Temporal Characteristics Document

We have defined two change operations, as listed in Table 1. In the following, we choose to present only the effect of the CreateTemporalCharacteristicDocument(TCD.json) change operation. The contents of the TCD.json file after the application of such an operation is as follows:

```
{ "temporalCharacteristicSet":{ } }
```

Table 1 Change operations acting on the whole temporal characteristic document

Change operation	Description
CreateTemporalCharacteristicDocument(TCD.json)	It creates a valid empty TCD. The argument is the name of the JSON file where the new TCD is stored.
DropTemporalCharacteristicDocument(TCD.json)	It removes the TCD.json file from the disk, with the constraint that the argument represents an empty TCD (i.e. like the one above initially created by CreateTemporalCharacteristicDocument). Any other contents must have been removed before.

4.4 Operations Common to the Temporal Logical and Physical Characteristics

These change operations can be applied either to the "logical" or to the "physical" container. We have defined six change operations; we provide them in Table 2.

Table 2 Change operations common to the temporal logical and physical characteristics

Change operation	Description
AddInclude(TCD.json, inWhat, temporalCharacteristicLocation)	It adds a new object as a new item, with specified "temporalCharacteristicLocation", to the "include" array in the inWhat (i.e., "logical" or "physical") container.
DeleteItemFromInclude(TCD.json, inWhat, temporalCharacteristicLocation)	It removes the item with specified "temporalCharacteristicLocation" from the "include" array in the inWhat (i.e., "logical" or "physical") container.
ChangeItemInInclude(TCD.json, inWhat, oldTemporalCharacteristicLocation, newTemporalCharacteristicLocation)	It changes the value of the property "temporalCharacteristicLocation" of the item with specified "oldTemporalCharacteristicLocation" to the value "newTemporalCharacteristicLocation", in the "include" array in the inWhat (i.e., "logical" or "physical") container.

Change operation	Description
AddDefaultTimeFormat(TCD.json, toWhat, plugin, granularity, calendar, properties, valueSchema)	It adds the "defaultTimeFormat" property with specified plugin, granularity, calendar, properties, and valueSchema to the toWhat (i.e., "logical" or "physical") container.
DeleteDefaultTimeFormat(TCD.json, fromWhat)	It removes the "defaultTimeFormat" property from the fromWhat (i.e., "logical" or "physical") container.
SetDefaultTimeFormat(TCD.json, inWhat, plugin, granularity, calendar, properties, valueSchema)	It changes the plugin, granularity, calendar, properties, or valueSchema of the "defaultTimeFormat" property in the inWhat (i.e., "logical" or "physical") container.

Here, we choose to present only the effect of the AddInclude change operation. The contents of the TCD.json file after the application of such an operation is as follows:
If (inWhat == logical) Then
TCD.json:

```
{ "temporalCharacteristicSet":{
   "logical":{
    "include":[
      "temporalCharacteristicLocation":"temporalCharacteristicLocation"
    ]
   },
   "physical":{ … } } }
```

Else
TCD.json:

```
{ "temporalCharacteristicSet":{
   "logical":{ },
   "physical":{
    "include":[
     "temporalCharacteristicLocation":"temporalCharacteristicLocation"
    ] } } }
```

4.5 Operations Specific to the Temporal Logical Characteristics

These change operations can be applied to the "logical" container only. We have identified twenty-two change operations of this kind; we provide them in Table 3. Here, we choose to present only the effect of the AddLogicalItem change operations. The contents of the TCD.json file after the application of such an operation is as follows:

```
{ "temporalCharacteristicSet":{
     "logical":{
        "logicalItems":[ { "target":"logicalItemTarget" } ]
     },
     "physical":{ … } } }
```

Table 3 Change operations specific to the temporal logical characteristics

Change operation	Description

Change operation	Description
AddLogicalItem(TCD.json, logicalItemTarget)	It adds a new item with specified "logicalItemTarget" to the "logicalItems" array in the "logical" container.
DeleteLogicalItem(TCD.json, logicalItemTarget)	It removes the item with specified "logicalItemTarget" from the "logicalItems" array in the "logical" container.
AddValidTimeToLogicalItem(TCD.json, logicalItemTarget, validTimeKind, validTimeContent, validTimeExistence, validTimeFrequency)	It adds the "validTime" property with specified "validTimeKind", "validTimeContent", "validTimeExistence" and "validTimeFrequency" to the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container. Notice that possible values of the three last arguments are as follows: • validTimeKind: either "state"or "event". • validTimeContent: either "constant" or "varying". • validTimeExistence: one of "constant", "varyingWithGaps", "varyingWithoutGaps".
DeleteValidTimeFromLogicalItem(TCD.json, logicalItemTarget)	It removes the "validTime" property from the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
SetValidTimeInLogicalItem(TCD.json, logicalItemTarget, validTimeKind, validTimeContent, validTimeExistence, validTimeFrequency)	It changes the "kind", "content", "existence" and/or "frequency" properties of the "validTime" property of the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
AddContentVaryingApplicabilityToValidTimeInLogicalItem(TCD.json, logicalItemTarget, contentVaryingApplicabilityBegin, contentVaryingApplicabilityEnd)	It adds a new object as a new item with specified "contentVaryingApplicabilityBegin" and "contentVaryingApplicabilityEnd" to the "contentVaryingApplicabilities" array, in the "validTime" property of the item specified with "logicalItemTarget" in the "logicalItems" array in the "logical" container.
DeleteContentVaryingApplicabilityFromValidTimeInLogicalItem(TCD.json, logicalItemTarget, contentVaryingApplicabilityBegin, contentVaryingApplicabilityEnd)	It removes the item with specified "contentVaryingApplicabilityBegin" and "contentVaryingApplicabilityEnd" from the "contentVaryingApplicabilities" array, in the "validTime" property of the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
ChangeContentVaryingApplicabilityInValidTimeInLogicalItem(TCD.json, logicalItemTarget, oldContentVaryingApplicabilityBegin, oldContentVaryingApplicabilityEnd, newContentVaryingApplicabilityBegin, newContentVaryingApplicabilityEnd)	It changes the value of the "begin" and/or the "end" property of the item with specified "oldContentVaryingApplicabilityBegin" and "oldContentVaryingApplicabilityEnd" in the "contentVaryingApplicabilities" array in the "validTime" property of the item with specified "logicalItemTarget" in the "logicalItem" array in the

Change operation	Description
	"logical" container.
SetMaximalExistenceInValidTimeInLogicalItem(TCD.json, logicalItemTarget, maximalExistenceBegin, maximalExistenceEnd)	It adds or changes the "maximalExistence" property, with specified "maximalExistenceBegin" and "maximalExistenceEnd", to the "validTime" property of the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
DeleteMaximalExistenceFromValidTimeInLogicalItem(TCD.json, logicalItemTarget, maximalExistenceBegin, maximalExistenceEnd)	It removes the "maximalExistence" property, with specified "maximalExistenceBegin" and "maximalExistenceEnd", from the "validTime" property of the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
AddTransactionTimeToLogicalItem(TCD.json, logicalItemTarget, transactionTimeKind, transactionTimeContent, transactionTimeExistence)	It adds the "transactionTime" property, with specified "transactionTimeKind", "transactionTimeContent", and "transactionTimeExistence", to the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container. Notice here that the four last arguments are optional.
DeleteTransactionTimeFromLogicalItem(TCD.json, logicalItemTarget)	It removes the "transactionTime" property from the item with specified "logicalItemTarget" in the "logicalItem" array in the "logical" container.
SetTransactionTimeInLogicalItem(TCD.json, logicalItemTarget, transactionTimeKind, transactionTimeContent, transactionTimeExistence)	It changes the "kind", "content", and/or "existence" of the "transactionTime" property of the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
AddItemIdentifierToLogicalItem(TCD.json, logicalItemTarget, itemIdentifierName, itemIdentifierTimeDimension)	It adds the "itemIdentifier" property with specified "itemIdentifierName" and "itemIdentifierTimeDimension" to the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container. Possible values of the "itemIdentifierTimeDimension" argument are: validTime, transactionTime, or bitemporal; default is validTime.
DeleteItemIdentifierFromLogicalItem(TCD.json, logicalItemTarget)	It removes the "itemIdentifier" property from the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
SetItemIdentifierInLogicalItem(TCD.json, logicalItemTarget, itemIdentifierName, itemIdentifierTimeDimension)	It changes the "name", and/or "timeDimension" of the "itemIdentifier" property in the item with specified "logicalItemTarget" in the "logicalItems" array in the "logical" container.
AddKeyrefToItemIdentifier(TCD.json, logicalItemTarget, keyrefName,keyrefType)	It adds an object as a new item of the array "keyrefs", with specified "keyrefName" and "keyrefType", to the "itemIdentifier" property of the item with

Change operation	Description
	specified logicalItemTarget of the "logicalItems" array in the "logical" container.
DeleteKeyrefFromItemIdentifier(TCD.json, logicalItemTarget, keyrefName)	It removes the item with specified keyrefName in "keyrefs" array from the "itemIdentifier" property of the item with specified logicalItemTarget of the "logicalItems" array in the "logical" container
ChangeKeyrefInItemIdentifier(TCD.json, logicalItemTarget, oldkeyrefName, newkeyrefName, oldkeyrefType, newkeyrefType)	It changes the value of refName property and/or the value of the refType property of the item specified with oldkeyrefName, in the "keyrefs" array in the "itemIdentifier" property of the item with specified "logicalItemTarget" in the "logicalItem" array in the "logical" container.
AddFieldToItemIdentifier(TCD.json, logicalItemTarget, fieldPath)	It adds an object as a new item of the "fields" array, with specified "fieldPath", to the "itemIdentifier" property of the item with specified "logicalItemTarget" in the "logicalItem" array in the "logical" container.
DeleteFieldFromItemIdentifier(TCD.json, logicalItemTarget, fieldPath)	It removes the item, with specified "fieldPath", from the "fields" array in the "itemIdentifier" property of the item with specified "logicalItemTarget" in the "logicalItem" array, in the "logical" container.
ChangeFieldInItemIdentifier(TCD.json, logicalItemTarget, oldFieldPath, newFieldPath)	It changes the item, with specified "oldFieldPath", to "newFiledpath", in the "fields" array in the "itemIdentifier" property of the item with specified "logicalItemTarget" in the "logicalItem" array, in the "logical" container.

4.6 Operations Specific to the Temporal Physical Characteristics

These change operations can be applied to the "physical" container only. We have defined ten change operations, which are listed in Table 4. Here, we choose to present only the effect of the AddStamp change operations. The contents of the TCD.json file after the application of such an operation is as follows:

```
{ "temporalCharacteristicSet":{
     "logical":{ … },
     "physical":{
        "stamps":[ { "target":"physicalStampTarget",
                     "dataInclusion":"stampDataInclusion",
                     "stampKind":{
                        "timeDimension":"stampKindTimeDimension",
                        "stampBounds":"stampKindStampBounds"
                     } } ] } } }
```

Table 4 Change operations specific to the temporal physical characteristics

Change operation	Description

Change operation	Description
AddStamp(TCD.json, stampTarget, stampDataInclusion, stampKindTimeDimension, stampKindStampBounds)	It adds an object as a new item of the "stamps" array, with specified "stampTarget" and "stampDataInclusion", to the "physical" container. Moreover, it adds to this item the "stampKind" property with specified "stampKindTimeDimension", and "stampKindStampBounds". Possible values of some arguments are as follows: • "stampDataInclusion": one of expandedEntity, referencedEntity, expandedVersion, or referencedVersion. • "stampKindTimeDimension": one of validTime, transactionTime, or bitemporal. • "stampKindStampBounds": either step or extent.
DeleteStamp(TCD.json,stampTarget)	It removes the item with specified "stampTarget" from the "stamps" array in the "physical" container.
SetDataInclusionInStamp(TCD.json, stampTarget, stampDataInclusion)	It introduces or changes the "dataInclusion" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container.
SetStampKindInStamp(TCD.json, stampTarget, stampKindTimeDimension, stampKindStampBounds)	It introduces or changes the "timeDimension" and/or "stampBounds" of the "stampKind" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container.
SetFormatInStampKindInStamp(TCD.json, stampTarget, stampPlugin, stampGranularity, stampCalendar, stampProperties, stampValueSchema)	It adds or changes the "format" property, with specified "stampPlugin", "stampGranularity", "stampCalendar", "stampProperties", and "stampValueSchema", to the "stampKind" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container.
DeleteFormatFromStampKindInStamp(TCD.json, stampTarget)	It removes the "format" property from the "stampKind" property of the item, with specified stampTarget, in the "stamps" array in the "physical" container..
AddOrderByFieldToStamp(TCD.json, stampTarget, newOrderByField)	It adds a new item "newOrderByField" to the "fields" array of the "orderBy" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container.
DeleteOrderByFieldFromStamp(TCD.json, stampTarget, orderByField)	It removes the "field" property having the value "orderByField" from the "orderBy" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container. When the last "field" is removed, also the "orderBy" property container is removed. Here, "orderByField" is either the value of the property "dimension" of a "time" property, or the value of a "target" property in a "field" property of

Change operation	Description
	the "orderBy" property.
DeleteOrderByFieldFromStamp(TCD.json, stampTarget, orderByField)	It removes the item "orderByField" from the "fields" array of the "orderBy" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container. When the last item of "fields" is removed, also the "orderBy" property container is removed.
ChangeOrderByFieldInStamp(TCD.json, stampTarget, oldOrderByField, newOrderByField)	It changes the item having the value "oldOrderByField" in the "fields" array to the value "newOrderByField", in the "orderBy" property of the item, with specified "stampTarget", in the "stamps" array in the "physical" container.

5 Application Example

As a motivating application example, we consider an international IT company that uses a JSON repository for storing data concerning its employees. We assume that on January 15, 2018, the NSDBA have created the first version of the conventional JSON schema of the employees shown in Figure 2. In particular, each employee in this company is described by an SSN, a name, a title, and a salary. An example of a conventional JSON document, that is an instance of this JSON schema, is provided in Figure 3.

```
{ "$schema":"http://json-schema.org/draft-04/schema#",
  "id": "http://jsonschema.net",
  "type": "object",
  "properties":{
     "employees":{
        "id":"http://jsonschema.net/employees",
        "type": "array",
        "items":{
           "type":"object",
           "properties":{
              "employee":{
                 "type":"object",
                 "properties":{
                    "SSN":{"type":"string"},
                    "name":{"type":"string"},
                    "title":{"type":"string"},
                    "salary":{"type":"number"} },
                 "required":["SSN", "name", "title", "salary"] } },
           "required":["employee"] } } },
  "required": ["employees"] }
```

Figure 2 The first version of the conventional JSON schema of the employees (employeesConventionalSchema_V1.json) on January 15, 2018.

```
{ "employees":[
     { "employee":{ "SSN":"18-X01-657A",
                    "name":"Khalil",
                    "title":"Developer",
                    "salary":"3000" } } ] }
```

Figure 3 The first version of the conventional JSON document of the employees (employeesConventionalDocument_V1.json) on January 15, 2018.

Then, the NSDBA annotated this first version of the conventional JSON schema with some temporal logical and physical characteristics. As to logical characteristics, we assume that he/she decided to represent the contents of the "salary" property as varying in valid-time, in order to keep the history along valid time of the changes the salary of each employee undergoes. As to physical characteristics, we assume that he/she chose to add a transaction-time physical timestamp to the object "employee", which means that whenever any property of the object "employee" changes, the entire "employee" object is duplicated to represent a new temporal version while the previous version is retained for archival purposes. The first version of the temporal characteristics document associated to the conventional schema of the employees is shown in Figure 4.

```
{ "temporalCharacteristicSet":{
     "logical":{
        "logicalItems":[
           {"target":"$.properties.employees..employee.properties.salary",
            "validTime":{
                "kind":"state",
                "content":"varying",
                "existence":"constant" } } ] },
     "physical":{
        "stamps":[
           { "target":"$.properties.employees.items.properties.employee",
             "dataInclusion":"expandedVersion",
             "stampKind":{
                "timeDimension":"transactionTime",
                "stampBounds":"extent" } } ] } } }
```

Figure 4 The first version of the temporal characteristics document (employeesTemporalCharacteristics_V1.json) on January 15, 2018.

Finally, when the application of changes to the conventional JSON schema has been completed, the NSDBA asks the system to save his/her work. In response, the system creates the temporal JSON schema in order to provide the linking information between the conventional JSON schema and its corresponding temporal logical and physical characteristics (stored in the temporal characteristics document), as shown in Figure 5. Notice that the temporal schema is a standard JSON document that ties the conventional JSON schema, logical characteristics, and physical characteristics together. The temporal JSON schema in the τJSchema environment is the equivalent of the conventional JSON Schema in the non-temporal JSON environment. This document contains objects (i.e. "slice" objects) that associate a series of conventional JSON schema definitions with

logical and physical characteristics, along with the time span during which the association was in effect.

```
{ "temporalJSONSchema":{
     "conventionalJSONSchema":{
        "sliceSequence":[
           { "slice":{
                "location":"employeesConventionalSchema_V1.json",
                "begin":"2018-01-15" } } ] },
     "temporalCharacteristicSet":{
        "sliceSequence":[
           { "slice":{
                "location":"employeesTemporalCharacteristics_V1.json",
                "begin":"2018-01-15" } } ] } } }
```

Figure 5 The temporal JSON schema of the employees (employeesTemporalSchema.json) on January 15, 2018.

Furthermore, let us assume that on April 10, 2018, the NSDBA decided to keep the history of the salary of each employee along both transaction and valid times. Hence, he/she has to change the first version of the temporal characteristics document in order to modify the logical item related to the "salary" property by adding a "transactionTime" object. We assume that he/she also decided to add another physical timestamp (having a bitemporal kind) to the property "salary". The second version of the temporal characteristics document is shown in Figure 6. Thus, the temporal JSON schema is also updated by adding a new slice object related to this new version of the characteristics document, as shown in Figure 7. Changes are presented in purple bold type.

```
{ "temporalCharacteristicSet":{
     "logical":{
        "logicalItems":[
           {"target":"$.properties.employees..employee.properties.salary",
            "validTime":{
               "kind":"state",
               "content":"varying",
               "existence":"constant" },
            "transactionTime":{
               "kind":"state",
               "content":"varying",
               "existence":"constant" } } ] },
     "physical":{
        "stamps":[
           {"target":"$.properties.employees.items.properties.employee",
            "dataInclusion":"expandedVersion",
            "stampKind":{
               "timeDimension":"transactionTime",
               "stampBounds":"extent" } },
           {"target":"$.properties.employees..employee.properties.salary",
```

```
            "dataInclusion":"expandedVersion",
            "stampKind":{
               "timeDimension":"bitemporal",
               "stampBounds":"extent" } } ] } } }
```

Figure 6 The second version of the temporal characteristics document (employeesTemporalCharacteristics_V2.json) on April 10, 2018.

```
{ "temporalJSONSchema":{
    "conventionalJSONSchema":{
       "sliceSequence":[
          { "slice":{
               "location":"employeesConventionalSchema_V1.json",
               "begin":"2018-01-15" } } ] },
    "temporalCharacteristicSet":{
       "sliceSequence":[
          { "slice":{
               "location":"employeesTemporalCharacteristics_V1.json",
               "begin":"2018-01-15" } },
          { "slice":{
               "location":"employeesTemporalCharacteristics_V2.json",
               "begin":"2018-04-10" } } ] } } }
```

Figure 7 The temporal JSON schema of the employees (employeesTemporalSchema.json) on April 10, 2018.

The sequence of change operations that have been specified by the NSDBA and performed on the temporal JSON schema (employeesTemporalSchema.json, Figure 5) and on the first version of the temporal characteristics document (employeesTemporalCharacteristics_V1.json, Figure 4) in order to update the temporal JSON Schema (see Figure 7) and to produce the second version of the temporal characteristics document (employeesTemporalCharacteristics_V2.json, Figure 6), can be expressed as the body of following schema change transaction:

```
Begin Transaction
(i) CreateNewJSONDocumentVersion(
         "employeesTemporalCharacteristics_V2.json",
         "employeesTemporalCharacteristics_V1.json")
(ii) AddTransactionTimeToLogicalItem(
         "employeesTemporalCharacteristics_V2.json",
         "$.properties.employees..employee.properties.salary", state,
         varying, constant)
(iii) AddStamp("employeesTemporalCharacteristics_V2.json",
         "$.properties.employees..employee.properties.salary",
         expandedVersion, bitemporal, extent)
(iv) AddSliceToTemporalJSONSchema("employeesTemporalSchema.json",
         temporalCharacteristicSet, current,
         "employeesTemporalCharacteristics_V2.json")
Commit
```

Notice that the transaction time associated to the execution of the transaction above is April 10, 2018, which is used by the system as value of the property "begin" of the new "slice" object corresponding to the new temporal characteristics document version, in the temporal JSON schema file. The first operation of this schema change transaction has been used to facilitate the creation of a new version of the characteristics document. In fact, we assume that the operation "CreateNewJSONDocumentVersion(NewJDV.json, ExistingJDV.json)" initially generates a new JSON document version "NewJDV.json" from an existing one "ExistingJDV.json" as a copy of it. After that, "NewJDV.json" is updated by the other operations of the schema change transaction in order to obtain, in the end, a new JSON document version that is actually different from the initial one according to the schema change specifications. As far as the last operation is concerned, it allows the addition to the temporal JSON schema of a new "slice" object, in order to take into account the creation of the new temporal characteristics document version.

6 Related Work

In the literature on Big Data and NoSQL databases devoted to store such data, to the best of our knowledge, there is no work that has dealt either with versioning and temporal aspects. However, there are some works that have separately studied temporal Big Data or versioning of non-temporal Big Data. In the following, we briefly present them.

In his survey, Cuzzocrea (2015) presents the state-of-the-art on temporal aspects of big data management (e.g., spatio-temporal modeling of Big Data, change detection in temporally-evolving network Big Data), and (ii) provides several future research directions related to this topic (e.g., indexing temporal Big Data, in-memory processing engines for temporal Big Data management). Among these topics, "integration with NoSQL platforms" can be found and the author claims that NoSQL systems are the most suitable computational platforms for managing temporal Big Data.

Since NoSQL databases could be document-oriented, key-value-oriented, graph-oriented or column-oriented, we organize the works that are related to our paper into three sets, according to the type of NoSQL database in which the work has been done.

In document-oriented NoSQL databases, Monger et al. (2012) use four particular properties (i.e., StartTransactionTime, EndTransactionTime, StartValidTime, and EndValidTime) to represent and store bi-temporal data in a JSON document; they integrate temporal aspects into conventional JSON documents. Mehmood et al. (2017) propose an approach for modeling temporal aspects of sensor Big Data for the MongoDB NoSQL DBMS, without dealing with the versioning/evolution issue of such data. Furthermore, Scherzinger and her colleagues have significantly contributed to the research done on JSON schema evolution, in JSON-based document-oriented NoSQL data stores. Their papers (Scherzinger et al., 2013, 2015a, 2015b, 2016; Klettke et al., 2016; Haubold et al., 2017) are presented in the following.

Scherzinger et al. (2013) propose a set of five low-level operations for adding, modifying and removing properties of entities, in a JSON NoSQL database that supports schema evolution (i.e., where only the last version of the structure of each entity is maintained).

In (Scherzinger et al., 2015a), the authors describe a system prototype, named

Cleager, that supports the operations introduced in (Scherzinger et al., 2013) and propagates changes eagerly (i.e., changes to the structure of an entity are automatically propagated to all instances of this entity).

Scherzinger et al. (2015b) present an Ecplise plugin, named ControVol, that propagates schema changes to instances, in a lazy manner. In fact, it (i) detects changes to entities' structures (e.g., removal of a property of an entity) in the program source code that make them different from structures of entities that are already stored in the NoSQL data store, (ii) returns warnings to application developers, and (iii) provides solutions to overcome these warnings.

In (Scherzinger et al., 2016), the authors propose Datalution, a system that supports the primitives of Scherzinger et al. (2013) and implements the eager and lazy strategies for propagating schema changes . In this work, it has been shown that the lazy strategy is more efficient than the eager one.

Klettke et al. (2016) deal with scalability of NoSQL data stores with respect to both long-term schema evolution (i.e., considering chains of pending schema change operations that have to be executed together) and lazy migration of underlying (Big Data) instances. Notice that chains of operations occur when legacy entities written by an application which then underwent multiple modifications are finally accessed by the application. In particular, the authors propose a rule-based composition for chains of pending schema change operations. After that, they experimentally compare four different scalable implementation strategies for lazy schema evolution on top of MongoDB: lazy stepwise, lazy composed, predictive, and incremental data migration.

Haubold et al. (2017) propose ControVol Flex, an extension of the ControVol plugin (Scherzinger et al., 2015b). It allows NoSQL application developers to choose their data migration strategy (i.e., eager or lazy) when a schema evolution is applied. Moreover, it could provide a combination of the two strategies, that is starting an eager migration of data in the background, while lazily migrating legacy entities, if the application requests access and eager migration has not reached them yet.

With respect to all above mentioned related works, our work has not dealt with change propagation strategies. We are planning to consider this issue in our future works.

In key-value NoSQL databases, Felber et al. (2014) deal with non temporal versioning of instances in a distributed key-value store. More precisely, they study design options that are available for implementing a versioned distributed key-value store on top of a conventional one. Saur et al. (2016) propose an approach and a tool, named KVolve, for managing lazy schema evolution in a JSON-based key-value NoSQL database. The authors experimentally show that their tool reduces downtime during the schema evolution process.

In graph-oriented NoSQL databases, Castelltort and Laurent (2014) discuss the criteria that should be fulfilled by a system which allows managing and querying data history in temporal graph-oriented NoSQL databases. These criteria are as follows: history, non-intrusivity, time-independence, and pluggability.

The works that are more strictly related with the approach in this paper and in its preliminary version (Brahmia et al., 2018a) are our previous works (Brahmia et al., 2016), (Brahmia et al., 2017), and (Brahmia et al., 2019a, 2019b, 2019c). The present work extends them, as they did not take into account versioning of the temporal characteristics. In fact, in (Brahmia et al., 2016) we have presented the τJSchema framework in its initial formulation, without any attention to the schema versioning issue, whereas in (Brahmia et al., 2017) we have focused on the versioning of conventional

JSON schemas only, without considering the versioning of temporal characteristics. In (Brahmia et al., 2019a), we have shown how τJSchema could be used to manage (i.e., create and validate) time-varying JSON documents. In (Brahmia et al., 2019b), we have extended (Brahmia et al., 2017) by proposing high-level operations for changing τJSchema schema, in a multi-schemaversioning environment. In (Brahmia et al., 2019c), we have proposed an approach for managing implicit schema versioning in τJSchema, that is schema versioning which is driven by instance updates and not by schema changes.

7 Conclusion

In this paper, we have completed the proposal of a systematic and comprehensive approach for managing temporal and versioning aspects of JSON-oriented data through the τJSchema framework (Brahmia et al., 2016; Brahmia et al., 2017). Precisely, we have dealt with versioning of temporal logical and physical characteristics that are associated to a conventional JSON schema. This work completes our previously presented framework and extends its functionalities to a full support of temporal JSON schema versioning: with the proposed solutions, both conventional schemas and their temporal characteristics can be versioned in an integrated and consistent environment.

Furthermore, our approach has been designed in order to provide the following advantages: (i) being systematic, it avoids the complications that may otherwise arise in the presence of Big Data, where the management of temporal characteristics evolution without a systematic approach would rely on the skills of the NSDBA and his/her actions could produce patchy results with ad hoc modifications cobbled together, (ii) old temporal characteristics of Big Data as well as new ones are all kept in the same environment (i.e., changing current temporal characteristics of Big Data leads to their implicit versioning and not to their overwriting, as in classical approaches where versioning is not supported) and, consequently, can all be exploited for satisfying user/applications requirements including temporal querying and analytics.

Currently, we are developing a τJSchema-based system prototype on top of MongoDB (a document-oriented NoSQL DBMS supporting the management of JSON documents), in order to show the feasibility of our approach.

As a part of our future work, we will address the problem of querying temporal JSON instance versions in the presence of multiple JSON schema versions, within the τJSchema framework. In particular, we plan to extend in this direction the JSONiq query language (Florescu and Fourny, 2013) that supports querying conventional JSON documents under a single schema version. Design and deployment of suitable change propagation strategies (e.g., eager, lazy or variants) is an issue that will also be considered in our future work.

References

Brahmia, Z., Grandi, F., Oliboni, B. and Bouaziz, R. (2014) 'Schema Change Operations for Full Support of Schema Versioning in the τXSchema Framework', *International Journal of Information Technology and Web Engineering*, Vol. 9, No. 2, pp. 20-46.

Brahmia, Z., Grandi, F., Oliboni, B. and Bouaziz, R. (2015) 'Schema Versioning', in: Khosrow-Pour, M. (Ed.) *Encyclopedia of Information Science and Technology (3rd edition)*, IGI Global, Hershey, Pennsylvania, USA, pp. 7651-7661.

Brahmia, S., Brahmia, Z., Grandi, F. and Bouaziz, R. (2016) 'τJSchema: A Framework for Managing Temporal JSON-Based NoSQL Databases', in: *Proc. of the 27th International Conference on Database and Expert Systems Applications (DEXA'2016)*, Porto, Portugal, Part 2, pp. 167-181.

Brahmia, S., Brahmia, Z., Grandi, F., Bouaziz, R. (2017) 'Temporal JSON Schema Versioning in the τJSchema Framework', *Journal of Digital Information Management*, Vol. 15, No. 4, pp. 179-202.

Brahmia, S., Brahmia, Z., Grandi, F. and Bouaziz, R. (2018a) 'Managing Temporal and Versioning Aspects of JSON-based Big Data via the τJSchema Framework', in: *Proc. of the International Conference on Big Data and Smart Digital Environment (ICBDSDE'2018)*, Casablanca, Morocco, Studies in Big Data, Vol. 53, Springer Nature AG, pp. 27-39.

Brahmia, Z., Grandi, F., Oliboni, B. and Bouaziz, R. (2018b) 'Supporting Structural Evolution of Data in Web-Based Systems via Schema Versioning in the τXSchema Framework', in: Elçi, A. (Ed.), *Handbook of Research on Contemporary Perspectives on Web-Based Systems*, IGI Global, Hershey, PA, USA, 2018, pp. 271-307.

Brahmia, S., Brahmia, Z., Grandi, F. and Bouaziz, R. (2019a) 'A Disciplined Approach to Temporal Evolution and Versioning Support in JSON NoSQL Data Stores', in: Ma, Z. and Yan, L. (Eds.), *Emerging Technologies and Applications in Data Modeling and Processing*, IGI Global, Hershey, PA, USA, 2019, pp. 114-133.

Brahmia, Z., Brahmia, S., Grandi, F. and Bouaziz, R. (2019b) 'Versioning Schemas of JSON-based Conventional and Temporal Big Data through High-level Operations in the τJSchema Framework', *International Journal of Cloud Computing*, in press.

Brahmia, Z., Brahmia, S., Grandi, F. and Bouaziz, R. (2019c) 'Implicit JSON Schema Versioning Driven By Big Data Evolution in the τJSchema Framework', in: *Proc. of the 3rd International Conference on Big Data and Networks Technologies (BDNT'2019)*, Leuven, Belgium, Lecture Notes in Networks and Systems, Vol. 81, Springer Nature Switzerland AG, pp. 23-35.

Castelltort, A. and Laurent, A. (2014) 'Managing and Querying Historical NoSQL GraphDatabases: The HNTP Criteria', *International Journal of Research in Information Technology*, Vol. 2, No. 2, pp. 184-196.

Cattell, R. (2010) 'Scalable SQL and NoSQL Data Stores', *ACM SIGMOD Record*, Vol. 39, No. 4, pp. 2-27.

Chen, C.P. and Zhang, C.Y. (2014) 'Data-intensive applications, challenges, techniques

and technologies: A survey on Big Data', *Information Sciences*, Vol. 275, pp. 314-347.

Corbellini, A., Mateos, C., Zunino, A., Godoy, D. and Schiaffino, S.N. (2017) 'Persisting big-data: The NoSQL landscape', *Information Systems*, Vol. 63, pp. 1-23.

Currim, F., Currim, S., Dyreson, C.E. and Snodgrass, R.T. (2004) 'A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with τXSchema', in: *Proc. of the 9th International Conference on Extending Database Technology (EDBT 2004)*, Heraklion, Crete, Greece, pp. 348-365.

Cuzzocrea, A. (2015) 'Temporal Aspects of Big Data Management: State-of-the-Art Analysis and Future Research Directions', in: *Proc. of the 22nd International Symposium on Temporal Representation and Reasoning (TIME'2015)*, Kassel, Germany, pp. 180-185.

Davoudian, A., Chen, L. and Liu, M. (2018) 'A Survey on NoSQL Stores', *ACM Computing Surveys*, Vol. 51, No. 2, Article 40.

Felber, P., Pasin, M., Riviere, E., Schiavoni, V., Sutra, P., Coelho, F., Oliveira, R., Matos M. and Vilaça, R.M.P. (2014) 'On the Support of Versioning in Distributed Key-Value Stores', in: *Proc. of the 33rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2014)*, Nara, Japan, pp. 95-104.

Florescu, D. and Fourny, G. (2013) 'JSONiq: The History of a Query Language', *IEEE Internet Computing*, Vol. 17, No. 5, pp. 86-90.

Gudivada, V.N., Rao D. and Raghavan V.V. (2014) 'NoSQL Systems for Big Data Management', in: *Proc. of the 2014 IEEE World Congress on Services (SERVICES'2014)*, Anchorage, AK, USA, pp. 190-197.

Haubold, F., Schildgen, J., Scherzinger, S. and Deßloch, S. (2017) 'ControVol Flex: Flexible Schema Evolution for NoSQL Application Development', in: *Proc. of the 17th Conference on Database Systems for Business, Technology, and Web (BTW'2017)*, Stuttgart, Germany, pp. 601-604.

Information Resources Management Association (IRMA) (2016) *Big data: Concepts, methodologies, tools, and applications*, IGI Global, Hershey, PA, USA.

Internet Engineering Task Force (IETF) (2013a) *JSON Schema: core definition and terminology*, Internet-Draft, 31 January 2013. https://tools.ietf.org/html/draft-zyp-json-schema-04 (accessed: November 20, 2019)

Internet Engineering Task Force (IETF) (2013b) *JSON Schema: interactive and non interactive validation*, Internet-Draft, 1 February 2013. http://tools.ietf.org/html/draft-fge-json-schema-validation-00 (accessed: November 20, 2019)

Internet Engineering Task Force (IETF) (2014) *The JavaScript Object Notation (JSON) Data Interchange Format*, Internet Standards Track document, March 2014.

Khosla, P.K. and Kaur, A. (2018) 'Big Data Technologies', in: Mittal, M., Balas, V.E., Hemanth, D.J. and Kumar, R. (Eds.) *Data Intensive Computing Applications for Big Data*, IOS Press, Amsterdam, The Netherlands, pp. 28-55.

Klettke, M., Störl, U., Shenavai, M. and Scherzinger, S. (2016) 'NoSQL Schema Evolution and Big Data Migration at Scale', in: *Proc. of the 2016 IEEE International Conference on Big Data (BigData'2016)*, Washington DC, USA, pp. 2764-2774.

Mehmood, N.Q., Culmone, R. and Mostarda, L. (2017) 'Modeling temporal aspects of sensor data for MongoDB NoSQL database', *Journal of Big Data*, Vol. 4, paper 8.

Monger, M.D., Mata-Toledo, R.A. and Gupta, P. (2012) 'Temporal Data Management in NoSQL Databases', *Journal of Information Systems & Operations Management*, Vol. 6, No. 2, pp. 237-243.

Pokorný, J. (2013) 'NoSQL databases: a step to database scalability in web environment', *International Journal of Web Information Systems*, Vol. 9, No. 1, pp. 69-82.

Saur, K., Dumitras, T. and Hicks, M.W. (2016), 'Evolving NoSQL Databases Without Downtime', in: *Proc. of the 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME'2016)*, Raleigh, North Carolina, USA, pp. 166-176.

Scherzinger, S., Klettke, M. and Störl, U. (2013) 'Managing Schema Evolution in NoSQL Data Stores', in: *Proc. of the 14th International Symposium on Database Programming Languages (DBPL'2013)*, Riva del Garda, Trento, Italy.

Scherzinger, S., Klettke, M. and Störl, U. (2015a) 'Cleager: Eager Schema Evolution in NoSQL Document Stores', in: *Proc. of the 16th Conference on Database Systems for Business, Technology, and Web (BTW'2015)*, University of Hamburg, Hamburg, Germany, pp. 659-662.

Scherzinger, S., Cerqueusy, T. and Cunha de Almeida, E. (2015b) 'ControVol: A Framework for Controlled Schema Evolution in NoSQL Application Development', in: *Proc. of the 31st IEEE International Conference on Data Engineering (ICDE'2015)*, Seoul, South Korea, pp. 1464-1467.

Scherzinger, S., Sombach, S., Wiech, K., Klettke, M. and Störl, U. (2016) 'Datalution: a tool for continuous schema evolution in NoSQL-backed web applications', in: *Proc. of the 2nd International Workshop on Quality-Aware DevOps (QUDOS@ISSTA'2016)*, Saarbrücken, Germany, pp. 38-39.

Sharma, S., Tim, U.S., Wong, J., Gadia, S.K. and Sharma, S. (2014) 'A brief review on leading big data models', *Data Science Journal*, Vol. 13, pp. 138-157.

Sharma, S., Tim, U.S., Gadia, S.K., Wong, J., Shandilya, R. and Peddoju, S.K. (2015) 'Classification and comparison of NoSQL big data models', *International Journal of Big Data Intelligence*, Vol. 2, No. 3, pp. 201-221.

Snodgrass, R.T. (ed.), Ahn, I., Ariav, G., Batory, D.S., Clifford, J., Dyreson, C.E., Elmasri, R., Grandi, F., Jensen, C.S., Käfer, W., Kline, N., Kulkarni, K., Cliff Leung, T.Y., Lorentzos, N., Roddick, J.F., Segev, A., Soo, M.D. and Sripada, S.M. (1995) *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, Norwell, MA, USA.

Snodgrass, R.T., Dyreson, C.E., Currim, F., Currim, S. and Joshi. S. (2008) 'Validating quicksand: Temporal schema versioning in τXSchema', *Data and Knowledge Engineering*, Vol. 65, No. 2, pp. 223-242.

Tiwari, S. (2011) *Professional NoSQL*, John Wiley & Sons, Inc., Indianapolis, Indiana, USA.

Appendix 1

This appendix provides in Figure A1 the JSON Schema code of TCSchema, the schema of temporal characteristics documents.

```
{ "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "http://jsonschema.net",
  "type": "object",
  "title": "TCSchema",
  "required": [ "temporalCharacteristicSet" ],
  "properties": {
    "temporalCharacteristicSet": {
      "id": "#/properties/temporalCharacteristicSet",
      "type": "object",
      "properties": {
        "logical": {
          "id": "#/properties/temporalCharacteristicSet/properties/
                 logical",
          "type": "object",
          "properties": {
            "include": {
              "id": "#/properties/temporalCharacteristicSet/properties/
                     logical/properties/include",
              "type": "array",
              "items": {
                "type": "object",
                "required": [ "temporalCharacteristicLocation" ],
                "properties": {
                  "temporalCharacteristicLocation": {
                    "id": "#/properties/temporalCharacteristicSet/
                           properties/logical/properties/include/
                           properties/temporalCharacteristicLocation",
                    "type": "string" } } } },
            "defaultTimeFormat": {
              "id": "#/properties/temporalCharacteristicSet/properties/
                     logical/properties/defaultTimeFormat",
              "type": "object",
              "properties": {
                "format": {
                  "id": "#/properties/temporalCharacteristicSet/
                         properties/logical/properties/defaultTimeFormat/
                         properties/format",
                  "type": "object",
                  "properties": {
                    "plugin": {
                      "id": "#/properties/temporalCharacteristicSet/
                             properties/logical/properties/
                             defaultTimeFormat/properties/format/
                             properties/plugin",
                      "type": "string" },
                    "granularity": {
                      "id": "#/properties/temporalCharacteristicSet/
                             properties/logical/properties/
                             defaultTimeFormat/properties/format/
                             properties/granularity",
```

```
            "type": "string" },
          "calendar": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   defaultTimeFormat/properties/format/
                   properties/calendar",
            "type": "string" },
          "properties": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   defaultTimeFormat/properties/format/
                   properties/properties",
            "type": "string" },
          "valueSchema": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   defaultTimeFormat/properties/format/
                   properties/valueSchema",
            "type": "string" } } } } },
  "logicalItems": {
    "id": "#/properties/temporalCharacteristicSet/
           properties/logical/properties/logicalItems",
    "type": "array",
    "items": {
      "type": "object",
      "required": [ "target" ],
      "properties": {
        "target": {
          "id": "#/properties/temporalCharacteristicSet/
                 properties/logical/properties/logicalItems/
                 properties/target",
          "type": "string" },
        "validTime": {
          "id": "#/properties/temporalCharacteristicSet/
                 properties/logical/properties/logicalItems/
                 properties/validTime",
          "type": "object",
          "properties": {
            "kind": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/logical/properties/
                     logicalItems/properties/validTime/
                     properties/kind",
              "type": "string",
              "enum": [ "state", "event" ] },
            "content": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/logical/properties/
                     logicalItems/properties/validTime/
                     properties/content",
              "type": "string",
              "enum": [ "constant", "varying" ] },
            "existence": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/logical/properties/
                     logicalItems/properties/validTime/
                     properties/existence",
              "type": "string",
              "enum": [ "constant", "varyingWithGaps",
                        "varyingWithoutGaps" ] },
            "contentVaryingApplicabilities": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/logical/properties/
```

```
                    logicalItems/properties/validTime/
                    properties/contentVaryingApplicabilities",
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "begin": {
                      "id": "#/properties/
                             temporalCharacteristicSet/
                             properties/logical/properties/
                             logicalItems/properties/
                             validTime/properties/
                             contentVaryingApplicabilities/
                             properties/begin",
                      "type": "string" },
                    "end": {
                      "id": "#/properties/
                             temporalCharacteristicSet/
                             properties/logical/properties/
                             logicalItems/properties/
                             validTime/properties/
                             contentVaryingApplicabilities/
                             properties/end",
                      "type": "string" } } } },
              "maximalExistence": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/logical/properties/
                       logicalItems/properties/validTime/
                       properties/maximalExistence",
                "type": "object",
                "properties": {
                  "begin": {
                    "id": "#/properties/temporalCharacteristicSet/
                           properties/logical/properties/
                           logicalItems/properties/validTime/
                           properties/maximalExistence/
                           properties/begin",
                    "type": "string" },
                  "end": {
                    "id": "#/properties/temporalCharacteristicSet/
                           properties/logical/properties/
                           logicalItems/properties/validTime/
                           properties/maximalExistence/
                           properties/end",
                    "type": "string" } } },
              "frequency": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/logical/properties/
                       logicalItems/properties/validTime/
                       properties/frequency",
                "type": "string" } } },
          "transactionTime": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/logicalItems/
                   properties/transactionTime",
            "type": "object",
            "properties": {
              "kind": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/logical/properties/
                       logicalItems/properties/transactionTime/
                       properties/kind",
```

```
            "type": "string",
            "enum": [ "state", "event" ] },
          "content": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/transactionTime/
                   properties/content",
            "type": "string",
            "enum": [ "constant", "varying" ] },
          "existence": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/transactionTime/
                   properties/existence",
            "type": "string",
            "enum": [ "constant", "varyingWithGaps",
                      "varyingWithoutGaps" ] },
          "frequency": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/transactionTime/
                   properties/frequency",
            "type": "string" } } },
      "itemIdentifier": {
        "id": "#/properties/temporalCharacteristicSet/
               properties/logical/properties/logicalItems/
               properties/itemIdentifier",
        "type": "object",
        "properties": {
          "name": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/itemIdentifier/
                   properties/name",
            "type": "string" },
          "timeDimension": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/itemIdentifier/
                   properties/timeDimension",
            "type": "string",
            "enum": [ "validTime", "transactionTime",
                      "bitemporal" ],
            "default": "validTime" },
          "keyrefs": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/logical/properties/
                   logicalItems/properties/itemIdentifier/
                   properties/keyrefs",
            "type": "array",
            "items": {
              "type": "object",
              "required": [ "refName" ],
              "properties": {
                "refName": {
                  "id": "#/properties/
                         temporalCharacteristicSet/
                         properties/logical/properties/
                         logicalItems/properties/
                         itemIdentifier/properties/
                         keyrefs/properties/refName",
                  "type": "string" },
                "refType": {
```

```
                        "id": "#/properties/
                               temporalCharacteristicSet/
                               properties/logical/properties/
                               logicalItems/properties/
                               itemIdentifier/properties/
                               keyrefs/properties/refType",
                        "type": "string",
                        "enum": [ "snapshot",
                                  "itemIdentifier" ] } } } },
                "fields": {
                  "id": "#/properties/temporalCharacteristicSet/
                         properties/logical/properties/
                         logicalItems/properties/itemIdentifier/
                         properties/fields",
                  "type": "array",
                  "items": {
                    "type": "object",
                    "required": [ "path" ],
                    "properties": {
                      "path": {
                        "id": "#/properties/
                               temporalCharacteristicSet/
                               properties/logical/properties/
                               logicalItems/properties/
                               itemIdentifier/properties/fields/
                               properties/path",
                        "type": "string" } } } } } } } } } },
"physical": {
  "id": "#/properties/temporalCharacteristicSet/properties/
         physical",
  "type": "object",
  "properties": {
    "include": {
      "id": "#/properties/temporalCharacteristicSet/properties/
             physical/properties/include",
      "type": "array",
      "items": {
        "type": "object",
        "required": [ "temporalCharacteristicLocation" ],
        "properties": {
          "temporalCharacteristicLocation": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/include/
                   properties/temporalCharacteristicLocation",
            "type": "string" } } } },
    "defaultTimeFormat": {
      "id": "#/properties/temporalCharacteristicSet/properties/
             physical/properties/defaultTimeFormat",
      "type": "object",
      "properties": {
        "format": {
          "id": "#/properties/temporalCharacteristicSet/
                 properties/physical/properties/
                 defaultTimeFormat/properties/format",
          "type": "object",
          "properties": {
            "plugin": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/
                     defaultTimeFormat/properties/
                     format/properties/plugin",
              "type": "string" },
```

```
            "granularity": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/
                     defaultTimeFormat/properties/
                     format/properties/granularity",
              "type": "string" },
            "calendar": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/
                     defaultTimeFormat/properties/
                     format/properties/calendar",
              "type": "string" },
            "properties": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/
                     defaultTimeFormat/properties/
                     format/properties/properties",
              "type": "string" },
            "valueSchema": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/
                     defaultTimeFormat/properties/
                     format/properties/valueSchema",
              "type": "string" } } } } },
    "stamps": {
      "id": "#/properties/temporalCharacteristicSet/properties/
             physical/properties/stamps",
      "type": "array",
      "items": {
        "type": "object",
        "required": [ "target", "stampKind" ],
        "properties": {
          "target": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/stamps/
                   properties/target",
            "type": "string"
          },
          "dataInclusion": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/stamps/
                   properties/dataInclusion",
            "type": "string",
            "enum": [ "expandedEntity", "referencedEntity",
                      "expandedVersion", "referencedVersion" ] },
          "stampKind": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/stamps/
                   properties/stampKind",
            "type": "object",
            "properties": {
              "timeDimension": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/stamps/
                       properties/stampKind/properties/
                       timeDimension",
                "type": "string",
                "enum": [ "validTime", "transactionTime",
                          "bitemporal" ] },
              "stampBounds": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/stamps/
                       properties/stampKind/properties/
```

```
                stampBounds",
            "type": "string",
            "enum": [ "step", "extent" ] },
          "format": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/stamps/
                   properties/stampKind/properties/
                   format",
            "type": "object",
            "properties": {
              "plugin": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/
                       stamps/properties/stampKind/
                       properties/format/properties/plugin",
                "type": "string"
              },
              "granularity": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/
                       stamps/properties/stampKind/
                       properties/format/properties/
                       granularity",
                "type": "string" },
              "calendar": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/
                       stamps/properties/stampKind/
                       properties/format/properties/
                       calendar",
                "type": "string" },
              "properties": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/
                       stamps/properties/stampKind/
                       properties/format/properties/
                       properties",
                "type": "string" },
              "valueSchema": {
                "id": "#/properties/temporalCharacteristicSet/
                       properties/physical/properties/
                       stamps/properties/stampKind/
                       properties/format/properties/
                       valueSchema",
                "type": "string" } } } } },
      "orderBy": {
        "id": "#/properties/temporalCharacteristicSet/
               properties/physical/properties/stamps/
               properties/orderBy",
        "type": "object",
        "properties": {
          "fields": {
            "id": "#/properties/temporalCharacteristicSet/
                   properties/physical/properties/stamps/
                   properties/orderBy/properties/fields",
            "type": "array",
            "items": {
              "id": "#/properties/temporalCharacteristicSet/
                     properties/physical/properties/stamps/
                     properties/orderBy/properties/fields/
                     items",
              "type": "object",
```

```
"oneOf": [
  {
    "properties": {
      "target": {
        "id": "#/properties/
               temporalCharacteristicSet/
               properties/physical/
               properties/stamps/
               properties/orderBy/
               properties/fields/
               items/properties/target",
        "type": "string" } } },
  {
    "properties": {
      "time": {
        "id": "#/properties/
               temporalCharacteristicSet/
               properties/physical/
               properties/stamps/
               properties/orderBy/
               properties/fields/
               items/properties/time",
        "type": "object",
        "required": [ "dimension" ],
        "properties": {
          "dimension": {
            "id": "#/properties/
                   temporalCharacteristicSet/
                   properties/physical/
                   properties/stamps/
                   properties/orderBy/
                   properties/fields/
                   items/properties/
                   time/properties/
                   dimension",
            "type": "string",
            "enum": [ "validTime",
                      "transactionTime",
                      "bitemporal" ]
          } } } } } ] } } } } } } } } } } } } } }
```

Figure A1 TCSchema: The JSON Schema for the specification of temporal logical and physical characteristics.