A Context Data Metamodel for Distributed Middleware Platforms in Smart Cities

Júlio Suzuki Lopes
Federal Institute of Paraíba (IFPB)
João Pessoa, Brazil
email: julio.lopes@ifpb.edu.br

Lucas Vale F. da Silva, Gledson Elias
Federal University of Paraíba (UFPB)
João Pessoa, Brazil
email: lucasfaustino@ppgi.ci.ufpb.br, gledson@ci.ufpb.br

***Abstract*—The concept of smart cities is related to the development of services, systems and applications to provide sustainable solutions for a huge and fast-growing population in urban areas. In a smart cities context, the wide range of application domains leads to a variety of large independent local repositories with non-unified data models that support very limited interoperability and, more importantly, hinder data reuse, integration, extension and partitioning. In order to address such issues, this paper presents a metamodel for specification and instantiation of context data in smart cities driven by interoperable distributed middleware platforms, enabling data integration, reuse, extension and partitioning, supplied by several independent data providers across a lot of application domains. Supported by an experimental prototype implementation, empirical results based on a semi-real dataset evince the potential benefits and practical applicability of the proposed metamodel.**

Keywords-smart cities; context modeling; context-awareness; data integration and partitioning.

I. INTRODUCTION

The concept of smart cities has gained a lot of attention from researchers around the world [1]. The core of this concept has explored the fact that the adoption of ICTs (Information and Communication Technologies) can improve quality of life and mitigate urban issues resulted from rapid population growth [2]. Among such ICTs, IoT (Internet of Things) is a key technology to solve major problems faced by people living in cities, enabling a range of services and applications by interconnecting digital and physical things (e.g., smartphones, TVs, vehicles) to share data and resources [3].

Nowadays, in large cities, a number of services, systems and applications have been developed with focus on specific urban problem domains, for instance, traffic and waste management, smart health and smart education [4]. Most of such software solutions are developed and managed by several public or private stakeholders, which adopt different ICT platforms and infrastructures [5], leading to a variety of large independent, local repositories with non-unified data models [6] and segmented data [5], resulting in the formation of the *information island* phenomenon [7]. Consequently, current solutions for smart cities support very limited interoperability and, more importantly, hinder data integration, reuse, extension and partitioning.

As a means to avoid information islands, in a way similar to the concept of virtual data warehouses [8], one of the main challenges in distributed middleware platforms for smart cities is to find a way to provide an integrated view of big urban data, enabling the development of interoperable services, systems and applications that communicate with each other for creating holistic and contextualized views of the cities [9][10].

In such a scenario, the adoption of a unified data model plays an important role, acting as a kind of glue that can bind services, systems and applications together. However, a unified data model for data integration is not enough. Regarding the dynamic, elastic and changeable nature of big urban data, such a unified data model ought also to facilitate data reuse, extension and partitioning.

In order to address such issues, this paper presents a metamodel for specification and instantiation of context data associated to all kinds of entities in smart cities. More importantly, the proposed metamodel, called DCDS (*Distributed Context Data Schema*), can be adopted as a unified data model in interoperable distributed middleware platforms for enabling data integration, reuse, extension and partitioning, supplied by several independent data providers across a lot of application domains. Supported by an experimental prototype implementation, empirical results based on a semi-real dataset evince the potential benefits and practical applicability of the proposed metamodel.

The remainder of the paper is structured as follows. Section 2 identifies the requirements related to data models for smart cities. Then, Section 3 discusses some related work, highlighting how identified requirements are handled by each one. Section 4 presents the proposed metamodel, detailing how to create schemas and instances related to smart cities entities, whose context data can be integrated, reused, extended and partitioned. Next, Section 5 shows a use case based on a large semi-real dataset on public urban transport. Concluding, Section 6 presents some final remarks, limitations and future work.

II. REQUIREMENTS FOR SMART CITIES DATA MODELS

This section identifies some requirements for data models that aim to ease the development of interoperable services, systems and applications in the smart cities scenario. To do that, an initial list of requirements has been derived through a literature review, covering a reasonable set of studies and proposals [11]-[20]. Note that the goal is not to be comprehensive, but rather to provide an overview of the main requirements. Accordingly, the conducted literature review has identified the following requirements:

 ISBN: 978-1-61208-715-3

- **Flexibility** – enables easy adaptation to different smart cities contexts, being not bounded to a single application or even a specific domain [13].
- **Expressivity** – concerned with the generic problem of knowledge representation [11], ensures a wide data design space for specification and instantiation of several data types related to smart cities entities.
- **Simplicity** – adopts a minimal number of structuring, composition and control rules, making more intelligible and easier data modelling processes [12].
- **Semanticability** – attaches semantic annotations as a means to enable human or automatic inspection and transformation of context data in heterogeneous distributed shared scenarios [14][15].
- **Granularability** – represents the characteristics of the context data at different levels of detail [13], including composite entities, structured datatypes and partial attribute assignments.
- **Interoperability** – supports structured unambiguous schemas to define entities and their attributes [17], which are helpful in data exchange among heterogeneous distributed platforms.
- **Reusability** – boosted also by structured schemas that act as reusable data contracts, encouraging to develop data-driven services in which ecosystem's actors can publish and share reusable datasets [18].
- **Integrability** – represents a step beyond reuse, in which ecosystem's actors can integrate other different external datasets, providing added value services or adapting to different target purposes [18].
- **Extensibility** – allows schemas and their associated instances to evolve over time and accommodate changes readily, dealing with the inherent diversity and dynamism of smart cities [19].
- **Partitionability** – enables context data related to smart cities entities to be partitioned or splitted up in multiple hosting nodes [20], supporting concurrent access to increase performance and scalability [16].

III. RELATED WORK

Data models have been proposed for platforms, systems, services and applications in several smart cities contexts. However, most of them have been adopted in centralized approaches, in which a single module, service, repository or node, herein called broker, is responsible for storing and managing the whole urban data, which obviously do not scale very well. Inversely, the proposed metamodel can be adopted in distributed approaches, in which multiple brokers can store context data related to smart cities entities in a distributed and even partitioned manner.

Based on XML (Extensible Markup Language), *ContextML* (CML) is a language designed by the C-Cast project [21] and adopted in the IoT architecture proposed by Cippra [22]. In both, it has been adopted as a common representation for exchanging context data between their respective components. Thus, it defines a markup language for context representation and mainly communication that should be supported by all components of compliant architectures. As a result, CML can meet the interoperability requirement, however, has a not so simple syntax. Note that the C-Cast project [21] and the Cippra architecture [22] propose centralized approaches that do not deal with issues related to partitionability.

Other two projects, *SENSEI* [23] and *IoT-A* [12], define data models to provide interoperability. Context entities are called resources in SENSEI and virtual entities in IoT-A. In both, the respective data models allow to represent real-world entities making possible to be aware of the context or environment in which such entities operate or can be accessed. Due to that, SENSEI and IoT-A can adequately meet the expressivity requirement, however, do not adopt a simple way to represent such entities, requiring expertise in low-level protocols, device standards and data formats. As a result, the effort for modelling entities is quite high and full of challenges even in simple cases. Despite that, both projects propose methods to orchestrate IoT services in order to combine together several resources or virtual entities in different granularities, providing high-level services based on semantic and ontological concerns.

NGSI (Next Generation Service Interfaces) [24][25] defines a context management information model that adopts the concept of entities as virtual representation of all kinds of real-world physical objects. In NGSI, each entity has a state represented by attributes, providing context-awareness in a simple, flexible and expressive way to compliant platforms and applications [12][24]. Despite defining mechanisms to manage references to external entities [25], in essence, NGSI presupposes the adoption of centralized approaches in compliant middleware platforms. As a result, a reasonable effort is required to explore partitioned context data among multiple brokers, imposing an issue related to scalability.

Based on the requirements identified for smart cities data models, Table I contrasts CML, SENSEI, IoT-A, NGSI and the proposed DCDS metamodel. The comparison takes as a starting point the evaluation presented by Jara *et al.* [12] and Nitti *et al.* [26] but enriched with new requirements and proposals. In all cases, the evaluated proposals were analyzed based on a set of concepts, properties and observations, directly extracted by us from their respective documentations. In Table I, note that the conducted evaluation adopts black dots to represent the degree of attendance for each proposal with respect to each requirement, varying from zero to three dots.

TABLE I. SMART CITY DATA MODELS

	CML	SENSEI	IoT-A	NGSI	DCDS
Flexibility	●●●	●●●	●●●	●●●	●●●
Expressivity	●●●	●●●	●●●	●●●	●●●
Simplicity	●	●	●	●●●	●●●
Semanticability	●	●●●	●●●	●●	●●
Granularability	●	●●	●●	●●	●●●
Interoperability	●●	●●	●●	●●	●●●
Reusability	●	●●	●●	●●	●●●
Integrability	●	●●	●●	●●	●●●
Extensibility	●●●	●●	●●	●●	●●●
Partitionability	-	-	-	●	●●●

As can be observed in Table I, among related work, NGSI has higher levels of attendance for almost all requirements, representing a promising data model to deal with smart cities entities. In fact, due to that, some existing projects have adopted the NGSI model, including the *Fiware* platform [27] that has gained a lot of attention in smart cities research communities. Despite that, as can be noticed, NGSI has limitations related to granularability, reusability, integrability, extensibility and partitionability. Regarding that smart cities scenarios require the ability to deal with massive urban data produced by a very large number of data sources and providers, such NGSI limitations have direct and strong impact on compliant smart cities platforms in respect to their scalability, usability and so practical applicability. Acting as a complementary approach, taking NGSI capabilities as a basis, the proposed DCDS metamodel introduces some simple but key additional features in order to better deal with several issues related to granularability, reusability, integrability, extensibility and partitionability.

IV. A DISTRIBUTED CONTEXT DATA SCHEMA

In order to improve existing proposals for smart cities data models, this section presents a context data metamodel, called DCDS, which provides the means to specify and instantiate context information associated to all kinds of real-world entities in a broad range of smart cities domains. As the main benefits and contributions, DCDS can be adopted as a unified data model in interoperable distributed middleware platforms for enabling data integration, reuse, extension and partitioning, supplied by several independent data providers across a lot of application domains. The remainder of this section describes the DCDS metamodel in more depth, including how to specify and instantiate smart cities entities, and how to support data partitioning.

A. Specifying and Instantiating Entities

DCDS adopts the concepts of *entity schema* and *entity instance* for representing the unambiguous single specification and the multiple associated instances for each context entity, respectively. Using UML (Unified Modeling Language), as illustrated in Figure 1, the representation of an *entity schema* has four constituting model elements.

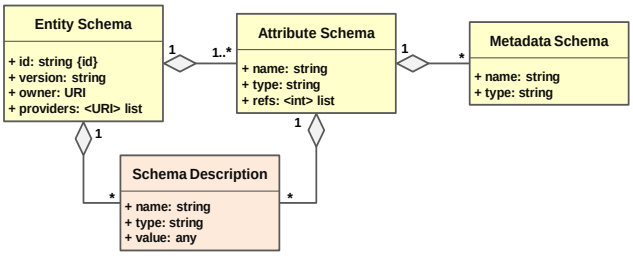

Figure 1. DCDS UML representation.

As the starting point of an entity schema, the *Entity Schema* model element has the following terms: ***id*** – provides a globally unique identifier for each entity schema; ***version*** – indicates the version of the respective entity schema, enabling providers to manage the entity schema lifecycle; ***owner*** – identifies the owner/provider that has specified the entity schema, allowing all providers to share schemas among them; and ***providers*** – defines a list of multiple providers in which related instances of the entity schema can be integrally or partially stored and retrieved.

Each context entity can have several context attributes, each one represented in the respective schema by the *Attribute Schema* model element, which has the following terms: ***name*** – provides a unique identifier for each attribute associated to the given entity schema; ***type*** – indicates the type of the respective attribute (e.g., string, integer and float); and ***refs*** – defines a list of references to providers in which the given attribute can be stored and retrieved. Note that, as explained later, the terms *providers* and *refs* are the basis for two distinct data partitioning types, which in turn leverage data granularity in an innovative way.

The other two model elements *Metadata Schema* and *Schema Description* are related to different types of optional metadata, which can be adopted to enrich information about schemas and instances, varying among simple textual descriptions, rich semantic annotations, and well-known adopted metrics, patterns and even standards. On the one hand, *Metadata Schema* denotes different types of metadata that can be associated to the given attribute during the instantiation of context entities. On the other hand, *Schema Description* represents metadata descriptions that can be associated to the specification of the own schema and the respective attributes. Each *Metadata Schema* and *Schema Description* model element has two terms, ***name*** and ***type***, providing a unique identifier for each metadata and indicating the type of the given metadata, respectively. Besides, each *Schema Description* has the ***value*** term for representing the specific metadata content.

Based on entity schemas, DCDS provides a simple, flexible and expressive way to describe unambiguous context entities in smart cities scenarios. As another important feature, each entity schema can be evaluated by compliant parsers and even engines to syntactically and semantically validate distributed context entity data, making possible to interoperate and integrate reusable context data managed by different providers in smart cities scenarios.

A step further, versioned schemas leverage extensibility, enabling to evolve context entity specifications and their instances over time, accommodating changes as a means to deal with the dynamic nature of urban data. For instance, new attributes and metadata can be included in schemas and subsequently their instances. Besides, already existing attributes and metadata can be renamed, updated or even removed. Therefore, it is possible to support CRUD (Create, Read, Update and Delete) operations for context entity schemas and instances. More importantly, such operations can also be employed in the list of providers where related instances and their attributes can be integrally or partially stored and retrieved. Of course, a compliant platform ought to define an API (Application Programming Interface) for dealing with the extensibility related to entity schemas and their attributes.

 ISBN: 978-1-61208-715-3

In practice, the DCDS UML representation must be converted to a context representation language. Among existing languages, for instance XML, CSV (Comma-Separated Values) and RDF (Resource Description Framework), JSON (JavaScript Object Notation) [28] has gained more and more attention in IoT scenarios, since it is simpler, smaller, faster and more readable than XML [29]. Figure 2 shows an example of a DCDS JSON schema.

```
{
"id":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"providers":["dcds.provider1.br", "dcds.provider2.br", "dcds.provider3.br"],
"schema-description":{
   "description":{
      "type":"text",
      "value":"Buses of the public transport system"}},
"location":{
   "type":"point",
   "refs":[1, 2],
   "schema-description":{
      "format":{
         "type":"text",
         "value":"GeoJson georeferenced location"}}},
"speed":{
   "type":"double",
   "refs":[1, 3],
   "metadata-schema":{
      "unit":{
         "type":"string"}}},
"people-ammount":{
   "type":"integer",
   "refs":[1, 2, 3]}
}
```

Figure 2. DCDS JSON schema representation.

In smart cities platforms, systems and applications, each real-world physical entity (e.g., sensors, actuators, automobiles and users) can be modeled and represented as a virtual context entity, which is denoted as an *entity instance* in the DCDS metamodel. Using a UML representation, as illustrated in Figure 3, an entity instance has three constituting model elements. Note that each entity instance must be compliant with its respective entity schema. As such, an entity instance can only have attributes and metadata previously specified in its corresponding entity schema.

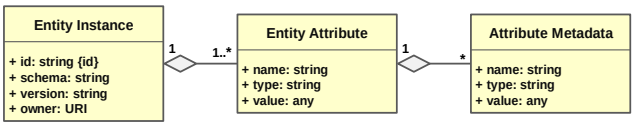

Figure 3. Context entity UML model.

Acting as the starting point of an entity instance, the *Entity Instance* model element has the following terms: ***id*** – provides a local identifier for each entity instance from the producer viewpoint; ***schema*** – indicates the compliant entity schema that defines the context information for the respective type of entity instance; ***version*** – denotes the version of the respective entity schema; ***owner*** – identifies the owner/provider that specified the compliant entity schema. Note that, together, the terms *id*, *schema*, *version* and *owner* provide a globally unique identifier for each entity instance, enabling compliant middleware platforms to provide naming services to leverage location transparency for schemas and their respective instances.

Each entity instance can have several context attributes, each one represented in the respective entity instance by the *Entity Attribute* model element, which has the following terms: ***name*** – denotes the attribute name defined in the corresponding entity schema; ***type*** – indicates the type of the respective attribute (e.g., string, integer and float); and ***value*** – represents the current value of the attribute in the given entity instance. In order to provide high granularity levels, a compliant middleware platform must have the capability of storing and retrieving integrally or partially the attributes of managed entity instances.

As specified in an entity schema, each entity instance can have optional associated metadata for providing contextual information about the given attribute instance. To do that, DCDS adopts the *Attribute Metadata* model element, which has three terms, ***name***, ***type*** and ***value***, denoting respectively the metadata name, type and specific content, all of them defined in the associated entity schema.

Figure 4 illustrates a simple example of a DCDS JSON instance, compliant with the DCDS JSON schema previously defined in Figure 2. As can be noticed, DCDS defines a context information model similar to NGSI [24] but enriched with the concept of entity schemas for leveraging the support to the identified requirements.

```
{
"id":"CT01-1 Circular-Tourism",
"schema":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"location":{
   "type":"point",
   "value":[30.52, 10.25]},
"speed":{
   "type":"double",
   "value":60.35,
   "unit":{
      "type":"string",
      "value":"kilometer per hour - km/h"}},
"people-ammount":{
   "type":"integer",
   "value":32}
}
```

Figure 4. DCDS JSON instance representation.

In order to provide a formal DCDS representation, EBNF (Extended Backus-Naur Form) grammars have been specified for entity schemas and instances, making possible to develop DCDS parsers. Due to space limitations and for the sake of simplicity, Figure 5 illustrates the EBNF grammar for entity schemas only, but without including the *Schema Description* model element.

```
dcds_schema ::= '{' entity_schema (',' attribute_schema)+ '}'

entity_schema ::= '"id":' '"' string '"' ','
                  '"version":' '"' string '"' ','
                  '"owner":' '"' uri '"' ','
                  '"providers":' '[' uri (',' uri)* ']'

attribute_schema ::= attribute_spec (',' metadata_schema)*

attribute_spec ::= '"' attr_name '"' ':' '{'
                   '"type":' '"' string '"' ','
                   '"refs":' '[' integer (',' integer)* ']'
                   '}'

metadata_schema ::= '"metadata-schema:"' '{'
                        '"' meta_name '"' ':' '{'
                            '"type":' '"' string '"' '}'
                    '}'

attr_name ::= string
meta_name ::= string
```

Figure 5. EBNF grammar for DCDS schemas.

B. Partitioning Context Instances and Attributes

Based on the DCDS metamodel, compliant middleware platforms can provide data partitioning in a two-fold perspective: *context instance partitioning* and *context attribute partitioning*. Such partitioning perspectives have a direct influence on requirements related to integrability, granularability and reusability, making possible services, systems and applications to integrate reusable context data from multiple providers in different granularity levels. Figure 6a and Figure 6b depict both partitioning approaches.

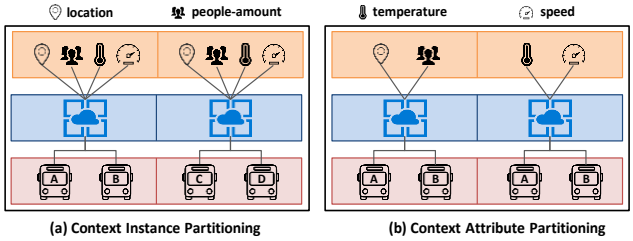

Figure 6. Data partitioning approaches.

In the *context instance partitioning* (Figure 6a), multiple independent data providers can store and manage subsets of entity instances related to the same entity schema. Thus, instead of storing the whole set of entity instances in a single provider, subsets of them are partitioned in multiple providers based on organizational, economical or geographical policies. Note that, for each entity instance, all currently valued attributes of the instance are integrally stored in a single provider, but without the need of assigning values to all attributes.

For example, in large cities, where there can be several bus operator companies responsible by providing the public transportation system, it sounds interesting that every company stores and manages context data about its own bus fleet. Figure 6a presents an example of instance partitioning, in which two providers manage all context data (*location*, *people-amount*, *speed* and *temperature*) associated with the bus fleet of two independent bus operator companies, which have the buses A/B and C/D, respectively. To do that, the ***providers*** term of the respective entity schema must simply include the URI (Uniform Resource Identifier) list of the authorized providers. Figure 7 illustrates the instance representation for buses A and C, which are integrally stored in different data providers with all attributes.

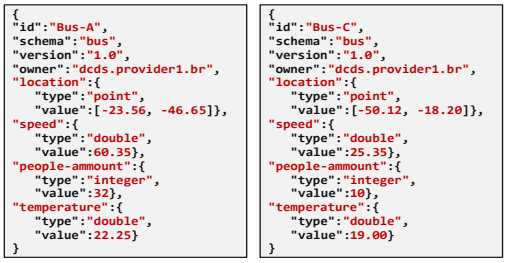

Figure 7. Instance partition example.

Differently, in the *context attribute partitioning* (Figure 6b), multiple independent data providers can store and manage subsets of attributes related to the same type of entity instances. That is, instead of storing all attributes of a given entity instance in a single provider, subsets of them are partitioned in multiple providers, probably based on expertise and capabilities of such providers. Again, there is no need of assigning values to all attributes.

For instance, in several cities, where the public transportation system is only provided by the municipal government, it seems interesting to hire distinct specialized companies for gathering and managing different context data types of interest. Figure 6b shows an example of attribute partitioning, in which two providers separately manage two distinct context attributes (*location*/*people-amount* and *speed*/*temperature*) associated with the municipal bus fleet. To do that, similarly, the ***providers*** term of the target entity schema must have the URI list of the authorized providers; but differently, for each attribute, the ***refs*** term must have the ordinal positions in the URI list of the providers that can store the respective attribute. Figure 8 illustrates the instance representation for bus A, which are partially stored in two different data providers without the need of assigning values to all attributes in each provider.

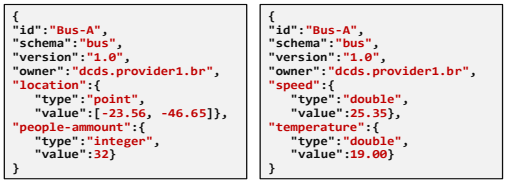

Figure 8. Attribute partition example.

In order to manage both partitioning perspectives, a compliant DCDS provider ought to adopt an integration process for combining all required instances associated to a given schema, version and provider. In the *context instance partitioning*, the requesting provider has to retrieve partial collections of instances stored in different providers and thereafter to integrate all them to provide the whole set of required instances. In the *context attribute partitioning*, the same process must be performed but, additionally, the requesting provider has to concatenate partial attributes of instances retrieved from different providers to reconstruct the whole set of attributes for all required instances.

It is important to emphasize that, in case of relational storage, both partitioning perspectives can be mapped to horizontal and vertical partitioning, as indicated in [30] for RDBMS (Relational Database Management Systems). In such a case, on the one hand, the *context instance partitioning* can be mapped to the *horizontal partitioning*, in which context instances are partitioned into disjoint sets of rows that are physically stored and accessed separately in different RDBMS-based providers. On the other hand, the *context attribute partitioning* can be mapped to the vertical partitioning, in which context attributes are partitioned into disjoint sets of columns that are physically stored and accessed separately in different RDBMS-based providers.

 ISBN: 978-1-61208-715-3

V. EXPERIMENTAL EVALUATION

In order to evaluate DCDS, based on HTTP (Hypertext Transfer Protocol) and REST (Representational State Transfer), a prototype implementation of a compliant distributed middleware platform, called *Sirius*, has been developed as a set of RESTful services for integrating multiple data providers, supporting the development of services, systems and applications in a broad range of smart cities domains.

As illustrated in Figure 9, *Sirius* adopts a service-oriented architecture. The *Context Schema Manager* deals with the versioned lifecycle of entity schemas, including CRUD operations for schemas and their respective attributes and metadata. Taking such specified schemas as a basis, the *Context Instance Manager* coordinates communication among other distributed *Sirius* brokers for storing, retrieving and integrating instances, including CRUD operations for instances and their respective attributes and metadata. By implementing a query processor, the *Context Query Engine* deals with a simple query language that can be adopted to transparently access and integrate context data from multiple providers. In order to provide independence from low-level database technologies, each *Context Broker Adapter* acts as a translator among DCDS JSON representations and a given low-level representation model. Therefore, several *Context Broker Adapters* can provide a way to map or transform different structured or unstructured data models into DCDS schemas.

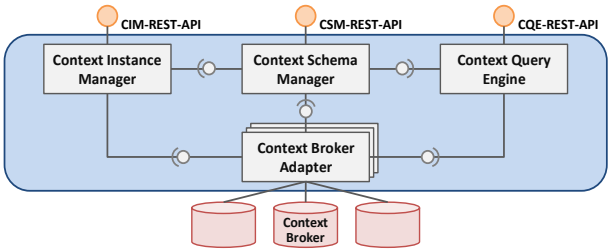

Figure 9. Sirius middleware platform.

Sirius was developed using *Flask*, a lightweight Python microframework for developing web applications and RESTful services. In addition, *Sirius* adopts *Orion Context Broker* [31] as a local broker for persisting instances. In order to define a distributed platform, virtual machines deployed in the AWS (Amazon Web Services) cloud infrastructure [32] act as context providers, running complete *Sirius* platform instances.

In the DCDS experimental evaluation, a semi-real dataset is adopted, taking as a basis the real-time database of the urban transport system provided by the São Paulo town hall in Brazil. Such a database provides an HTTP RESTful API to access the real-time context data related to all bus fleets provided by bus operator companies in São Paulo. Originally, the context database provides information related to the name of the line, location and travel direction of about 10,000 buses. However, for enriching the context attributes associated to buses, the conducted experiments have included additional synthetic attributes. Thus, in the bus schema, each bus instance has the following attributes: *location*, *people-amount*, *temperature* and *speed*.

In a way similar to Figure 6a and Figure 6b, two different experiments have been successfully evaluated using six virtual machines in the AWS cloud infrastructure, four of them to deploy *Sirius* instances, another one to deploy *Orion* and the last one to use as a client to dispatch operations. In the first experiment, the *context instance partitioning* approach was evaluated, spreading 10,000 bus instances in four *Sirius* brokers, which means around 2,500 bus instances in each broker. In such a case, each bus instance has all four attributes stored in the respective broker. In the second experiment, the *context attribute partitioning* approach was evaluated, storing all 10,000 bus instances repeatedly in each *Sirius* broker. However, differently, each bus instance has only one attribute stored in each *Sirius* broker. As a mean to define a comparing baseline, the experiment also has a third configuration in which all buses are stored a centralized *Orion* broker.

As an initial performance evaluation, a configurable set of concurrent users, varying from 10 to 100, dispatch read requests for each configuration of the experiment, each one recovering the whole set of 10,000 stored instances. Then, for each set of concurrent users, the total time for processing such requests was measured. The experiment was conducted using the *JMeter* load testing tool [33], which allows to configure different load profiles and calculate their respective response times for a variety of services. In *JMeter*, HTTP requests can be modeled as users, which perform concurrent requests to target services.

Figure 10 shows the total response time for each configuration (*instance partitioning*, *attributed partitioning* and *centralized Orion*) and the set of concurrent users (*10*, *20*, *40*, *60*, *80*, *100*). Of course, the lower the response time, the better performance has the evaluated configuration.

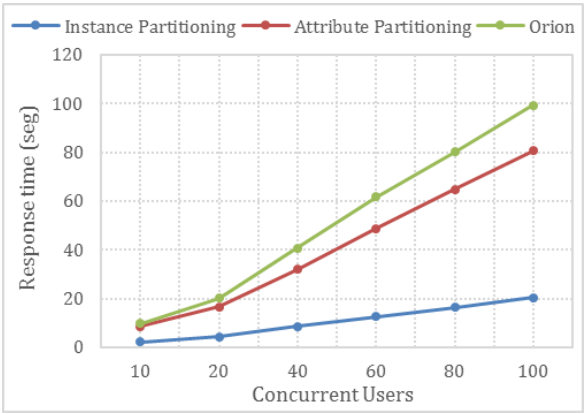

Figure 10. Performace evaluation.

As can be noticed, in both partitioning configurations, as the number of concurrent users increases, the *Sirius'* distributed approach provides better and better performance when contrasted with the *Orion's* centralized approach. For example, considering 100 concurrent users, the gains of the instance and attribute partitioning approaches are around 79.4% and 18.9%, respectively.

 ISBN: 978-1-61208-715-3

VI. CONCLUSION AND FUTURE WORK

In order to evolve from not so scalable, centralized smart cities platforms, this paper presents the DCDS metamodel, which provides a means to specify and instantiate distributed, partitioned and versioned context information associated to all kinds of real-world entities in a broad range of smart cities domains. As the main contributions, DCDS can be adopted as a unified data model in interoperable, scalable and distributed middleware platforms, enabling the development of services, systems and applications that can easily deal with capabilities related to data integration, reuse, extension and partitioning, supplied by several independent providers across a lot of smart cities domains.

On the one hand, in DCDS, versioned schemas and instances enable to evolve entity schemas and their instances over time, accommodating changes imposed by dynamic urban data. On the other hand, DCDS enables data partitioning in an innovative two-fold approach. First, multiple providers can manage subsets of instances related to the same schema. Second, multiple providers can manage subsets of attributes related to the same type of instances. As contributions, such versioning and partitioning capabilities leverage requirements related to extensibility, integrability, granularability and reusability, enabling services, systems and applications to integrate reusable, extensible context data from multiple providers in distinct granularity levels.

It is important to highlight that both partitioning approaches have the potential to provide better response time and scalability in compliant middleware platforms due to gains imposed by parallel access [16], concentrating frequently accessed instances or attributes in providers with more available or less overloaded processing power, communication bandwidth and storage capacity.

Despite such contributions, from the viewpoint of complaint platforms, DCDS does not have concerns related to security requirements, such as access control, authentication, confidentiality and denial of service. Besides, in collaborative distributed smart cities initiatives, business models ought to be adopted in order to regulate how to monetize data providers. Together, concerns related to security and business models represent key potential branches for future work.

As another future branch, it is important to conduct performance, load and stress tests with much bigger scenarios, including a lot of data provider nodes and several types of context entities in different smart cities domains. Such tests enable to gather more confidence about empirical findings related to enumerated requirements of interest, but also including scalability and availability.

REFERENCES

[1] T. Nam and T. A. Pardo, "Conceptualizing Smart City with Dimensions of Technology, People, and Institutions", 12th Int. Conf. on Digital Government Research, 2011, pp. 282-291.

[2] H. Chourabi, *et al.*, "Understanding Smart Cities: An Integrative Framework", 45th Hawaii Int. Conf. on Syst. Sci., 2012, pp. 2289-2297.

[3] E. Borgia, "The Internet of Things Vision: Key Features, Applications and Open Issues", Computer Commun., vol. 54, pp. 1-31, Dec. 2014.

[4] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris, "Smarter Cities and their Innovation Challenges", Computer, Issue 6, pp. 32-39, Jun. 2011.

[5] N. Ben-Sassi, *et al.*, "Service Discovery and Composition in Smart Cities", Int. Conf. on Adv. Inf. Syst. Eng., 2018, pp. 39-48.

[6] A. J. Jara, *et al.*, "Smart Cities Semantics and Data Models", Int. Conf. on Inf. Technol. & Syst., 2018, vol. 721, pp. 77-85.

[7] F. J. Villanueva, M. J. Santofimia, D. Villa, J. Barba, and J. C. Lopez, "Civitas: The Smart City Middleware from Sensors to Big Data", 7th Int. Conf. on Innovative Mobile and Internet Serv. in Ubiquitous Comput., 2013, pp. 445-450.

[8] M. Crowe, C. Begg, F. Laux, and M. Laiho, "Data validation for big live data", 9th Int. Conf. on Advances in Databases, Knowl., and Data Appl., 2017, pp. 30-36.

[9] I. A. Hashem, *et al.*, "The Role of Big Data in Smart City", Int. J. of Inf. Manag., vol. 36, n. 5, pp. 748-758, Oct. 2016.

[10] M. Strohbach, H. Ziekow, V. Gazis, and N. Akiva, "Towards a Big Data Analytics Framework for IoT and Smart City Applications", In: F. Xhafa, L. Barolli, A. Barolli, and P. Papajorgji (eds), Modeling and Processing for Next-Generation Big-Data Technologies, vol 4, pp. 257-282, 2015.

[11] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A Survey of Context Data Distribution for Mobile Ubiquitous Systems", ACM Comput. Surveys, vol. 44, n. 4, pp. 24-28, Aug. 2012.

[12] A. J. Jara, *et al.*, "Semantic Web of Things: An Analysis of the Application Semantics for the IoT Moving Towards the IoT Convergence", Int. J. of Web and Grid Serv., vol. 10, n. 2/3, pp. 244-272, Apr. 2014.

[13] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, "A Data-Oriented Survey of Context Models", ACM SIGMOD Record, vol. 36, n. 4, pp. 19-26, Dec. 2007.

[14] R. Reichle, *et al.*, "A Comprehensive Context Modeling Framework for Pervasive Computing Systems", Int. Conf. on Distrib. Appl. and Interoperable Syst., vol.5053, 2008, pp. 281-295.

[15] A. J. Jara, Y. Bocchi, D. Fernandez, G. Molina, and A. Gomez, "An Analysis of Context-Aware Data Models for Smart Cities: Towards Fiware and ETSI CIM Emerging Data Model", Int. Archives of Photogrammetry, Remote Sens. and Spatial Inf. Sci., vol. XLLII-4/W3, pp. 43-50, Set. 2017.

[16] M. Boussard, *et al.*, "A Process for Generating Concrete Architectures", Enabling Things to Talk, Springer, pp. 45-111, 2013.

[17] W. C. McGee, "On User Criteria for Data Model Evaluation", ACM Trans. on Database Syst., vol. 1, n. 4, pp. 370-387, 1976.

[18] A. A. García, M. O. U. Criado, and C. P. Heredero, "The Ecosystem of Services around Smart Cities: An Exploratory Analysis", Procedia Comput. Sci., vol. 64, pp. 1075-1080, 2015.

[19] J. Lee, S. Baik, and C. C. Lee, "Building an Integrated Service Management Platform for Ubiquitous Ecological Cities", Computer, vol. 44, n. 6, pp. 56-63, 2011.

[20] R. White and J. Tantsura, "Navigating Network Complexity: Next-Generation Routing with SDN", Service Virtualization, and Service Chaining, Addison-Wesley, 2015.

[21] M. Knappmeyer, S. L. Kiani, C. Fra, B. Moltchanov, and N. Baker, "ContextML: A Light-Weight Context Representation and Context Management Schema", 5th Inter. Symp. on Wireless Pervasive Comput., 2010, pp. 367-372.

[22] M. R. Crippa, "Design and Implementation of a Broker for a Service-Oriented Context Management and Distribution Architecture", Undergraduate Thesis, UFRGS, Jul. 2010.

[23] V. Tsiatsis, *et al.*, "The SENSEI Real World Internet Architecture", In: Towards the Future Internet: Emerging Trends from Europe Research, pp. 247-256, 2010.

[24] OMA, "NGSI Context Management", version 1.0, May 2012.

[25] OMA, "NGSI Registration and Discovery", version 1.0, May 2012.

[26] M. Nitti, V. Pilloni, G. Colistra, and L. Atzori, "The Virtual Object as a Major Element of the Internet of Things: A Survey", IEEE Commun. Surveys & Tuts., vol. 18, n. 2, pp. 1228-1240, Nov. 2016.

[27] Fiware, http://www.fiware.org [retrieved: April, 2019].

[28] JSON, http://www.json.org [retrieved: April, 2019].

[29] S. Zunke and V. D'Souza. "JSON vs XML: A Comparative Performance Analysis of Data Exchange Formats", Int. J. of Comp. Sci. and Netw., vol. 3, n. 4, pp. 257-261, Aug. 2014.

[30] S. Agrawal, V. Narasayya, and B. Yang, "Integrating vertical and horizontal partitioning into automated physical database design", Int. Conf. on Manag. of Data, 2004, pp. 359-370.

[31] Orion, http://fiware-orion.readthedocs.io [retrieved: April, 2019].

[32] AWS, http://aws.amazon.com/pt/ec2 [retrieved: April, 2019].

[33] JMeter, http://jmeter.apache.org [retrieved: April, 2019].

 ISBN: 978-1-61208-715-3