Weaving Open Services with Runtime Models for Continuous Smart Cities KPIs Assessment

Martina De Sanctis[1 [0000−0002−9417−660X]], Ludovico Iovino[1 [0000−0001−6552−2609]], Maria Teresa Rossi[1 [0000−0003−0273−7324]], and Manuel Wimmer[2 [0−0002−1124−7098]]

[1] Gran Sasso Science Institute, L'Aquila, Italy
{firstname.lastname}@gssi.it
[2] CDL-MINT, Johannes Kepler University, Linz, Austria
manuel.wimmer@jku.at

Abstract. The automatic Key Performance Indicators (KPIs) assessment for smart cities is challenging, since the input parameters needed for the KPIs calculations are highly dynamic and change with different frequencies. Moreover, they are provided by heterogeneous data sources (e.g., IoT infrastructures, Web Services, open repositories), with different access protocol. Open services are widely adopted in this area on top of open data, IoT, and cloud services. However, KPIs assessment frameworks based on smart city models are currently decoupled from open services. This limits the possibility of having runtime up-to-date data for KPIs assessment and synchronized reports. Thus, this paper presents a generic service-oriented middleware that connects open services and runtime models, applied to a model-based KPIs assessment framework for smart cities. It enables a continuous monitoring of the KPIs' input parameters provided by open services, automating the data acquisition process and the continuous KPIs evaluation. Experiment shows how the evolved framework enables a continuous KPIs evaluation, by drastically decreasing (∼88%) the latency compared to its baseline.

Keywords: Models@run.time · Continuous Monitoring · Smart Cities Assessment.

1 Introduction

The Smart Cities (SCs) ecosystem is an ideal ground for service-based applications, where the role of Service-Oriented Architecture (SOA) is to enable the integration between city services to realize innovative services and applications (e.g., [1, 2]). Particularly, we focus on the smart governance [3] process within SCs, concerning the use of technology in processing information and supporting *smart decision making*. Specifically, it exploits Key Performance Indicators (KPIs) assessment to measure qualitative metrics over cities to support their smart and sustainable growth[3]. For instance, the International Telecommunication Union (ITU) defined a list of all the KPIs for Smart Sustainable Cities, along with its collection methodology [4]. The KPIs assessment process involves different tasks, e.g., retrieving input data, calculating indicators, and reporting evaluation results. Traditional approaches, e.g., manual or spreadsheet-based approaches[4], envisage a significant human contribution to perform such operations, with expensive and repetitive activities requiring resources and time to be performed.

Enabling *automation* in the KPIs assessment to both the retrieval of input data and calculation of KPIs, is not trivial, since the input parameters needed for the calculations may come from different types of data sources (e.g., IoT infrastructure, open data

[3] https://bit.ly/3ekdT9D [4] https://bit.ly/37EFR9r

repositories or statistics elaborated by public entities) in different formats. Moreover, the values of the input parameters can change periodically (e.g., hourly), and thus, the KPIs assessment process has to be synchronized and re-assessed accordingly.

On the one hand, Web services and APIs, i.e., the most common way to specify open services in the SC domain, are widely adopted to build new applications on top of open data, IoT, and cloud services. On the other hand, model-based approaches are exploited in the SC domain, i.e., to represent complex systems through abstract models [5]. However, despite the huge availability of SC services and models, they are currently not well-connected, which would be required to reach the notion of a digital twin [6]. Moreover, the currently available frameworks for the KPIs calculation are mainly online spreadsheets[5], which are far from being automated, and Web-based applications (see, e.g., [7]) only providing a fixed set of predefined KPIs. In our previous work [8], we presented a flexible and automated model-based approach for KPIs assessment in SCs. However, the research efforts in [8] have focused on the definition of the model-based artifacts of the framework, while ignoring the relationship with the independent and heterogeneous data sources providing KPIs input parameters and the constant synchronization with them. Thus, in [8] the input parameters values have to be manually retrieved and updated in the models.

With these premises, we developed a generic service-oriented middleware that connects open services and runtime models. Specifically, we evolve the architecture in [9] of our approach in a service-oriented fashion, by means of the message-oriented middleware enabling: (i) continuous monitoring of KPIs input parameters from heterogeneous sources available as (open) services, and (ii) runtime models evolution with up-to-date input parameters for the SC modeling artifacts, in accordance with the real-world SCs evolution reflected by the open services. In other words, *we provide a generic solution for monitoring runtime model parameters from open services for SCs models*. Thus, we turn SC models into digital twins, by weaving open services and the runtime models, allowing the automated information flow from the system to the model [6].

The rest of the paper is organized as follows: background and motivation for this work are discussed in Section 2. Section 3 presents the proposed approach. Evaluation results of the implemented prototype are reported in Section 4. Finally, Section 5 discusses the related work and draws conclusions and future directions.

2 Background and Motivation

In this section, we describe the assessment process realized by the smart cities KPIs modeling framework from [8], its limitations, and the challenges we aim to address.

2.1 A Smart Cities KPIs Assessment Framework

Fig. 1 depicts the overall KPIs evaluation approach consisting in four main phases, each with dedicated input and output elements. The assessment of a SC starts from the *SC Modeling* phase, during which the city under evaluation is modeled, by means of MDE techniques. In the *SC Model*, SCs are designed in terms of their stakeholders (e.g., municipality), infrastructures (e.g., IoT infrastructures), data sources (e.g., open data, IoT services) and data types. This way, the *SC Model* provides the *input parameters* needed to calculate the KPIs of interest, as we will see in the following. In the *KPIs Definition* phase, by following *KPIs Guidelines/Documentation* (e.g., [4, 7]), the user models or select the relevant KPIs for the SC under evaluation (e.g., Air Pollution KPI, Travel Time Index KPI). In the *KPIs Model* given as output, the calculation formulae of the

[5] `https://bit.ly/3dT1zwV`

selected KPIs are defined by using a textual Domain-Specific Language (DSL) [10]. The designed *SC Model* and *KPIs Model*, are the inputs for an evaluation engine that executes the *KPIs Assessment* over the candidate SC. The assessment phase returns an *Evaluated KPIs Model* reporting the KPIs concrete values resulted from the assessment. The *Evaluated KPIs Model*, in turn, is the input of the *KPIs Visualization* phase during which *Dashboard*s representing the KPIs status are generated, through code generation.

In the following, we give a trivial but concrete example of a KPI evaluated for a smart city, as done in [8]. Specifically, we consider the KPI *Air Pollution (AP)* that measures the air quality based on the values reported for specific pollutants [4]. It is based on the *Air Quality Index (AQI)* formula, calculated as in (1), where p refers to the pollutant (e.g., $PM2.5$) whereas the *legal limit* is established by the law:

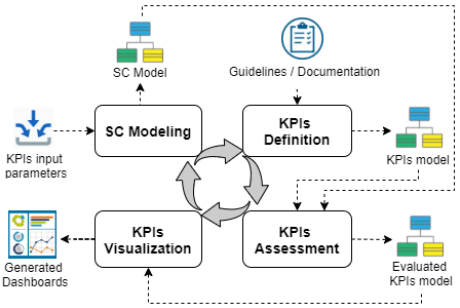

Fig. 1: KPIs assessment: process overview.

$$AQI_p = (\textit{measured concentration}_p / \textit{legal limit}) \times 100 \quad (1)$$

The worst AQI_p (i.e., the greater) determines the *Air Pollution AP* KPI, which is evaluated w.r.t. five evaluation classes, i.e., Excellent, Good, Discrete, Bad, Terrible. To calculate the AP KPI we need, as *input parameters*, the measured concentrations of the required pollutants. These values are provided by the *SC Model* together with the data sources from which they have been collected, e.g., the Breezometer open API[6]. An excerpt of the *SC model* of the smart city of *L'Aquila* is depicted in Fig. 2. It shows the tree-view of the model as shown in the Eclipse Modeling Framework (EMF)[7], on top of which the assessment framework has been designed. Fig. 2 also shows an ex-

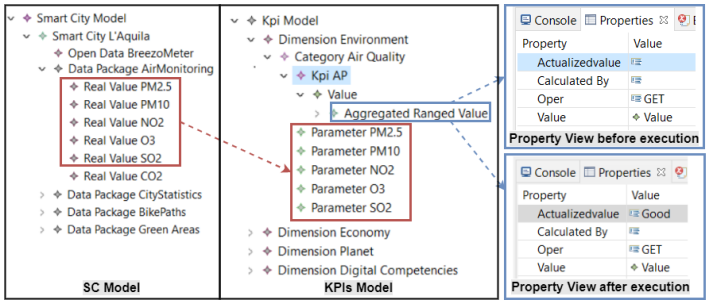

Fig. 2: Excerpts of the smart city model designing the city of *L'Aquila*.

cerpt of the tree-view of the *KPIs Model* where the AP KPI formula is nested into the `Aggregated Ranged Value`. The corresponding textual representation through the provided DSL [8] can be found in our online repository[8]. It is important to notice that, in the KPI definition, the names of the parameters whose values are provided by the *SC Model* must conform. This match will be executed by the evaluation engine during the assessment. Fig. 2 eventually shows the *Property View* reporting the AP KPI *Actualized Value before* and *after* the assessment, i.e., Good.

[6] `https://www.breezometer.com/`
[7] `https://bit.ly/31c1GHG`
[8] `https://github.com/iovinoludovico/runtime-kpi-assessment`

2.2 Limitation and Challenges

In the selected framework, the KPIs assessment process envisages the involvement of the users (e.g., KPIs experts) not only in the initial design of the required models but also in the manual retrieving of the KPIs input parameters. This means that the user has to manually fill the *SC model* every time the KPIs input parameters change, and then trigger the re-execution of the KPIs assessment process. These manual tasks are time-consuming and error-prone. Moreover, some KPIs input parameters are highly dynamic, since they change with very different frequencies (e.g., monthly, hourly). For instance, data about pollutants concentrations can be collected every hour or even minutes. Lastly, KPIs input parameters are provided by a multitude of heterogeneous sources, such as IoT sensors, social media, open data, usually available and accessible as (open) APIs and services. Given the discussion above, we can identify the following challenges: **(C1)** The framework must guarantee a *continuous and possibly automated monitoring* of KPIs parameters sources, i.e., services, and runtime update of the models using these parameters. **(C2)** The framework must provide *real-time evaluated KPIs*, i.e., as much up-to-date as possible w.r.t. the current status of the smart city, given that KPIs support and affect the decision-making processes in smart cities. For these reasons, we believe that our approach can benefit from a service-oriented continuous monitoring feature, providing automatic gathering of data and runtime models updates (i.e., *SC model*s), enabling the dashboards synchronization.

3 Runtime Model Updates by Continuous Monitoring

The architecture [9] behind our previous framework [8] was focused on the assessment phase, thus keeping the gathering of data and the consequent models update as manual tasks. To address the challenges discussed above and overcome the current limitations of the framework, we refactored and extended its architecture by adding a *message-oriented middleware* enabling continuous monitoring of KPIs data sources and runtime update of models in the KPIs assessment for SCs, as shown in Fig. 3. This extension evolves the manual and standalone framework into an automatic and service-oriented one, where heterogeneous data sources continuously feed the assessment process.

In the *front-end*, we have the KPIs Modeling Editor (implemented with Xtext[9]) devoted to the selection and definition of the relevant KPIs for the SC under evaluation, through custom textual DSLs. The Smart City Modeling Editor [11] (implemented with Sirius [12]), instead, helps users to model the SC under evaluation through the exploitation of graphical functionalities [11]. Lastly, the graphical Dashboard, obtained through model to code transformations (and visualized with Picto[10]) allows the interpretation of the KPIs assessment results.

In the *back-end*, the Requests Manager handles: (i) KPIs assessment requests to the Evaluation Engine (modeled with the Epsilon Object Language[11]) that is responsible for performing the SC evaluation; (ii) visualization requests from the Evaluation Engine to the Dashboard component. In particular, the Dashboard Synchronizer converts the KPIs model instantiated after the assessment in an HTML file, which is in sync with source files of the Dashboard. The synchronizer has its own listener that every time the model changes the HTML file is reloaded, updating the views. The Requests Manager also handles requests to the Models Manager to gather or store the models needed in the KPIs assessment process. The models manager handles the persistence of models in the SC Models Repository and the KPIs Models Repository.

[9] https://www.eclipse.org/Xtext/ [10] https://bit.ly/3l8jL9s
[11] https://www.eclipse.org/epsilon/doc/eol/

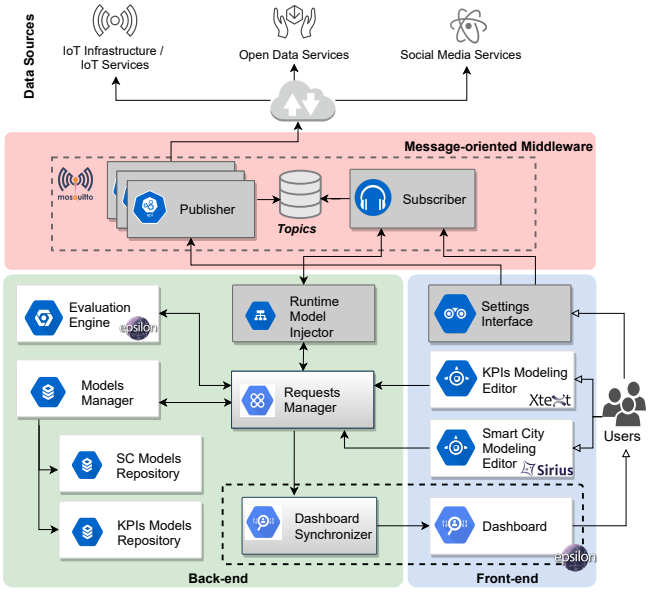

Fig. 3: The service-oriented architecture for the continuous KPIs assessment of SCs. The arrows shape the data-flow among components. White and blurred grey components show the architecture from [9], where the blurred grey ones required modifications for this work. In grey, instead, the newly integrated components.

We now describe the new components and the *message-oriented middleware*. The Publishers and the Subscriber implement the classic publish/subscribe communication pattern based on Topics. The type of topics are the types of parameters the SC can handle in the evaluation, e.g., pollutants, travel time, and so on. Specifically, the Publisher components can be multiple, considering the multiple data sources. They send calls to data sources, e.g., open services, to gather the input parameters needed for the KPIs calculations. Each publisher prepares the data, before publishing changed parameters on the assigned topic. These components are interfaces implemented, in turn, by specific Java classes.

The topics are published as MQTT [13] messages with a specific structure, namely $lat/\{latitude\}/long/\{longitude\}/parameter$, allowing multi-city evaluation. Latitude and longitude are the GPS coordinate of the smart city under evaluation. This way we can provide multiple publishers as types of data gatherer for different cities, by matching the smart cities coordinates. Then, the subscriber is able to distinguish which data intercept, by using the latitude and longitude of the smart city under evaluation. The Subscriber is devoted to the synchronization of the models with the data sources. It receives changed parameters through the subscription to the corresponding topic and actions can be triggered, as specified in the following. The Runtime Model Injector is invoked by the Subscriber, when it receives new data from the topics, and devoted to the retrieving of the SC models that need to be synchronised. It checks if the input parameters in the models are in line with the ones received by the Subscriber by querying the model. It is also in charge of SC models *update*, that is the operation of filling the models with the received up-to-date data, through EOL queries. Furthermore, the Runtime Model Injector interacts with the Requests Manager to trigger the KPIs assessment process due to changed input parameters, requesting the last saved SC models. The overall monitoring process is enabled by the user that is responsible for setting up the continuous monitoring features, only for the dynamic parameters with the $runtime$ attribute set to TRUE in the SC model, through the Settings Interface. It is used for defining the *topics* to which the Subscriber has to subscribe, and configure the APIs and the frequency with which they must be called in the Publishers. This setting is provided through a name based convention, so if the parameters in the SC model are included in the topics, then the Runtime Model Injector will consider them in the process.

It is worth noting that after converting the architecture of our framework in a service-oriented one, enabling continuous monitoring and runtime models update, the assessment approach can be enhanced in a models@runtime evaluation. Lastly, if new KPIs are modeled, to monitor their input parameters it suffices to add new publishers according to provided templates and connect them with parameters names in the *SC Model*.

4 Implementation and Evaluation

We demonstrate the evolved approach through a running example, by applying the extended evaluation framework over a real smart city, i.e., *L'Aquila*. Specifically, we consider 6 *dynamic* KPIs and four Publishers as data sources for the input parameters of these KPIs, i.e., 3 open services and an IoT infrastructure. Details can be found in our online repository[12]. We experimentally assess the service-based continuous KPIs assessment and compare it with its previous version where the KPIs parameters continuous monitoring and runtime models update were not available.

Research Question (RQ). We aim to answer the following RQ: What is the impact of KPIs input *data retrieving*, *models update* and *evaluation engine execution* tasks on the latency? How does the presented approach compare with its baseline [9] w.r.t. the impact to latency of these tasks?

Experiment Setup. To answer to **RQ**, we have conducted an experiment by comparing the *baseline framework* proposed in [9], requiring manual data retrieving and models filling, with the fully *automated framework*, enabling continuous monitoring and runtime models update presented in this paper. We consider one subject smart city and 6 dynamic KPIs. For this experiment, the framework run on a Macbook Pro 2019, 2,3 GHz 8-Core Intel Core i9 processor, 32 GB 2667 MHz DDR4 RAM and 2TB SSD of storage. This laptop runs the Mosquitto client, all the publishers, the subscriber and the evaluation engine. We run the experiment for 6 hour. Anytime (and only when) the *SC Model* is not up to date, the subscriber updates the parameters in the model with the new received values and it runs the evaluation engine that triggers the dashboard's update. All these activities have been automatically monitored and measured in a log.

Results for RQ. Table 1 shows to which extent the *data retrieving*, *SC Model update* and *evaluation engine* execution contribute to the framework latency, in both the Automated and Baseline Exps. Specifically, Table 1 reports the average execution time

Approach	Data Retrieving	SC Model Update	Evaluation Engine
Baseline	14.929,89 ms	18.668,28 ms	6.605,22 ms
Automated	525,89 ms	2,96 ms	138,82 ms

Table 1: Latency contributed by the three phases.

for each of the three phases in milliseconds (ms), for both Exps. As expected, each of the three phases requires quite more time in the Baseline Exp, where the data retrieving and model update (performed manually) are the most time consuming ones. Moreover, the latency in the Baseline Exp could also depend from communication issues among the users involved in the monitoring of data sources and update of the models. The evaluation engine execution also contributes more to the latency in the Baseline Exp, since it has to be manually launched anytime changes are applied in the *SC Model* due to the evolution of the KPIs input parameters.

Discussion. Current limitations concern the generalizability and scalability of the approach. Although it has been applied on a single smart city to evaluate 6 KPIs, it uses

[12] `https://github.com/iovinoludovico/runtime-kpi-assessment`

techniques (e.g., PubSub pattern, open APIs) that can be generalized and extended to more complex systems, as long as there are accessible data sources to get real input parameters for *dynamic* KPIs to be measured on real smart cities. Indeed, PubSub is known to offer better scalability w.r.t. traditional client-server, by means of parallel operation and message caching. Of course, the message-oriented middleware might add a network latency delay. However, keeping the data retrieving and update of models as manual tasks is impractical, considering the huge number of identified KPIs.

5 Related Work, Conclusion, and Future Work

Several SCs architectures can be found in the literature [14]. Matar *et al.* [15] present an approach for designing smart city's ecosystems, by means of a reference architecture (RA), *SmartCityRA*, by exploiting model-driven architecture techniques. Voronin *et al.* [16], propose an RA for designing a smart city context through the use of Big Data. However, both approaches [15, 16] do not support SCs evaluation.

The currently available *frameworks for the KPIs calculation* are still far from being automated. Manual and online spreadsheets are not appropriate for dynamic data retrieving. Among Web-based framework, Bosch *et al.* [7] select a set of KPIs to assess SCs to measure their smartness and to visualize them with graphical representations. However, the tool does neither envisage automatic calculation nor retrieving of data. Moustaka *et al.* [17] present a framework to support maturity benchmarking of SCs. However, it lacks continuous monitoring and automatic injection of data.

Run-time monitoring (RM) and *models@runtime* [18] have been widely exploited in model-based systems and applications. Hili *et al.* [19] propose an architecture supporting RM of executions of models of real-time and embedded systems. In their case studies they connect the code generated from a model with a range of external tools for different purposes (e.g., run-time verification). While they apply RM on model artifacts, we monitor heterogeneous third-party data sources, to dynamically update model artifacts (i.e., the *SC Model*). Other approaches exploiting RM are proposed in the IoT context, to support the management of its inherent heterogeneity, as done, for instance, in Chen *et al.* [20]. Differently, we aim to apply continuous monitoring both to IoT architectures and to other data sources (e.g., Open services), with diverse architectures and access protocols. This further increases heterogeneity beyond that inherent in IoT architectures. In service-based systems, Johng *et al.* [2] propose a continuous service monitoring framework to control changes in services by detecting SLA's violations, to facilitate collaborations among DevOps software teams. Differently, our framework is not just a notification system but integrates a complete SC assessment.

Nevertheless, despite the availability of numerous smart cities services, and the wide use of models@runtime and service-based technologies, to the best of our knowledge, it does not exist a service-oriented framework for the continuous evaluation of SCs.

In conclusion, we presented a service-oriented architecture for a model-based KPIs assessment framework supporting decision-making processes in SCs, and providing a robust and fully automated platform. As future work, we aim to integrate time-series databases [21] enabling temporal models to support the storing of historical values. This way, we may store the history of the input parameters and time-based analysis of the KPIs result, which can be used to visualize the evolution of the smart city and its performance over time. Lastly, we aim to deploy the framework online as a Web application. This way, the framework would become itself a smart city service provider.

Acknowledgements. This work was partially supported by the Centre for Urban Informatics and Modelling (CUIM) and the PON Cultural Heritage-AIM1880573, National Projects at

GSSI, and by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG).

References

1. A. Bucchiarone, M. De Sanctis, and A. Marconi, "ATLAS: A world-wide travel assistant exploiting service-based adaptive technologies," in *Proc. of the 15th Int. Conf. on Service-Oriented Computing (ICSOC)*. Springer, 2017, pp. 561–570.
2. H. M. Johng, A. Kalia, J. Xiao, M. Vukovic, and L. Chung, "Harmonia: A continuous service monitoring framework using devops and service mesh in a complementary manner," in *Service-Oriented Computing*, 2019, pp. 151–168.
3. D. Mutiara, S. Yuniarti, and B. Pratama, "Smart governance for smart city," *IOP Conf. Series: Earth and Environmental Science*, vol. 126, pp. 12–73, 2018.
4. International Telecommunication Union, "Collection Methodology for Key Performance Indicators for Smart Sustainable Cities," 2017, https://bit.ly/3vFsfqW.
5. F. Rosique, F. Losilla, and J. A. Pastor, "A domain specific language for smart cities," in *Proc. of the 4th Int. Electronic Conf. on Sensors and Applications*, 2018.
6. F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *Proc. of the 1st Int. Conf. on Systems Modelling and Management (ICSMM)*. Springer, 2020, pp. 43–54.
7. P. Bosch, S. Jongeneel, V. Rovers, H.-M. Neumann, M. Airaksinen, and A. Huovila, "Citykeys indicators for smart city projects and smart cities," 2017, https://bit.ly/3tr9WEt.
8. M. De Sanctis, L. Iovino, M. T. Rossi, and M. Wimmer, "MIKADO – A Smart City KPIs Assessment Modeling Framework," *Softw. Syst. Model.*, 2021.
9. M. De Sanctis, L. Iovino, M. T. Rossi, and M. Wimmer, "A flexible architecture for key performance indicators assessment in smart cities," in *Proc. of 14th European Conference on Software Architecture (ECSA)*, vol. 12292. Springer, 2020, pp. 118–135.
10. D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. Polack, "Requirements for domain-specific languages," in *Proc. of the Workshop on Domain-Specific Program Development*, 2006.
11. F. Basciani, M. T. Rossi, and M. De Sanctis, "Supporting smart cities modeling with graphical and textual editors." CEUR-WS.org, 2020.
12. V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of DSM graphical editor," in *Proc. of the Int. Conf. on Intelligent Engineering Systems*, 2014, pp. 233–238.
13. R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.
14. W. M. da Silva, A. Alvaro, G. H. R. P. Tomas, R. A. Afonso, K. L. Dias, and V. C. Garcia, "Smart cities software architectures: A survey," in *Proc. of the 28th Annual ACM Symposium on Applied Computing (SAC)*. ACM, 2013, p. 1722–1727.
15. M. Abu-Matar and R. Mizouni, "Variability modeling for smart city reference architectures," in *IEEE Int. Smart Cities Conf.*, 2018, pp. 1–8.
16. D. Voronin, V. Shevchenko, O. Chengar, and E. Mashchenko, "Conceptual big data processing model for the tasks of smart cities environmental monitoring," in *Digital Transformation and Global Society*. Springer, 2019, pp. 212–222.
17. V. Moustaka, A. Maitis, A. Vakali, and L. Anthopoulos, "CityDNA Dynamics: A Model for Smart City Maturity and Performance Benchmarking," in *Proc. of the 6th Int. Workshop: Web Intelligence and Smart Cities*, 2020.
18. N. Bencomo, S. Götz, and H. Song, "Models@run.time: a guided tour of the state of the art and research challenges," *Softw. Syst. Model.*, vol. 18, no. 5, pp. 3049–3082, 2019.
19. N. Hili, M. Bagherzadeh, K. Jahed, and J. Dingel, "A model-based architecture for interactive run-time monitoring," *Softw. Syst. Model.*, vol. 19, pp. 959–981, 2020.
20. X. Chen, A. Li, X. Zeng, W. Guo, and G. Huang, "Runtime model based approach to IoT application development," *Frontiers of Computer Science*, vol. 9, pp. 540–553, 2015.
21. A. Mazak, S. Wolny, A. Gómez, J. Cabot, and M. Wimmer, "Temporal Models on Time Series Databases," *Journal of Object Technology*, vol. 19, p. 3:1, 2020.