A Flexible Architecture for Key Performance Indicators Assessment in Smart Cities

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
{martina.desanctis,ludovico.iovino,mariateresa.rossi}@gssi.it
[2] CDL-MINT, Johannes Kepler University, Linz, Austria
manuel.wimmer@jku.at

Abstract.

1 Introduction

In the domain of smart cities, the *smart governance* concerns the use of technology in processing information and decision making enabling open, transparent and participatory governments [1], by also supporting the knowledge sharing among the involved actors. The main instrument through which smart governance operates is represented by *Key Performance Indicators* (KPIs) [2] representing raw set of values that can provide information about relevant measures that are of interest for understanding the progress of a smart city. The European Commission released and promoted the *Sustainable Development Goals* (SDGs[3]) to be achieved in 2020 [3], on top of which the International Telecommunication Union (ITU) drafted a list of all the KPIs for Smart Sustainable Cities (SSCs), along with its collection methodology [4].

However, decision making for smart cities through KPIs assessment is a quite challenging task. This is also due to the complexity of smart cities that are, de facto, *systems of systems* involving different dimensions (e.g., mobility, environment, education), each managed by different stakeholders, from public administrations to private institutions, that not always communicate with each other. Moreover, despite the support provided by Information and Communications Technologies (ICT) in managing different aspects of complex systems, such as smart cities (e.g., [5]), the currently available frameworks for the KPIs assessment are too rigid, not easy to suit each specific smart city's peculiarity, and often not released as open frameworks. Examples are online spreadsheets or Excel programs[4] embedding the used models and KPIs calculation formulas, or web/cloud-based frameworks with pre-defined set of computable KPIs, without considering that smart cities may differ in several aspects, based on their stage of economic development, available services, geographical implications. Moreover, KPIs can vary depending on the spatial granularity (e.g., small, medium and metropolitan cities).

In this context, we argue that a systematic methodology allowing smart cities stakeholders to easily *define*, *model* and *measure* the KPIs of interest for their cities, efficiently supporting the decision making process, is necessary. The methodology should

[3] `https://sustainabledevelopment.un.org/sdgs` [4] Key Performance Indicators in Power Pivot at `https://bit.ly/37EFR9r`

further support the KPIs *customization* and *evolution* to suit the unique features of different and heterogeneous smart cities. In this direction, we defined a model-based approach for the automatic KPIs assessment in smart cities as an effective instrument for smart governance enabling the stakeholders knowledge sharing, data interpretation and smart cities evaluation and comparison. The approach foresees the separate modeling of smart cities and KPIs by experts and stakeholders, leveraging Model-Driven Engineering (MDE) techniques [6]. The two models are then used by an evaluation engine that will provide the evaluated KPIs over the candidate smart cities.

To make the approach robust, a supportive software architecture is required, that needs to leverage the offered abstraction in order to keep deployment aspects outside of the box. In this paper, we present *a flexible architecture supporting the model-based approach for KPIs assessment in smart cities* that identifies both required and optional components and corresponding functionalities needed for realizing the automatic KPIs assessment approach, while showing two main *flexibility points* allowing for different specification of the architecture, thus of the overall methodology implementation. The flexibility is given by (i) *different deployment patterns* that can be followed for specifying the architecture (e.g., standalone, hybrid, online); (ii) the *technology-independent nature* of the shaped components, which enables the use of diverse technologies for implementing the designed architectural components, to also better suit the chosen deployment style (i.e., online modeling editors better suit in online deployment of the system). This last point also includes the tool-independent nature of the KPIs *evaluation engine*, which plays a central role in the overall methodology. The proposed architecture further benefits from all the positive aspects of the model-based nature of the approach, i.e., support to software evolution, automation of software production with code generation, support to technological bridges, and so on. Moreover, our architecture may provide guidelines for the definition of MDE tools (i.e., for the development, interpretation, transformation of models) where quality evaluation is the main objective [7].

2 Related Work

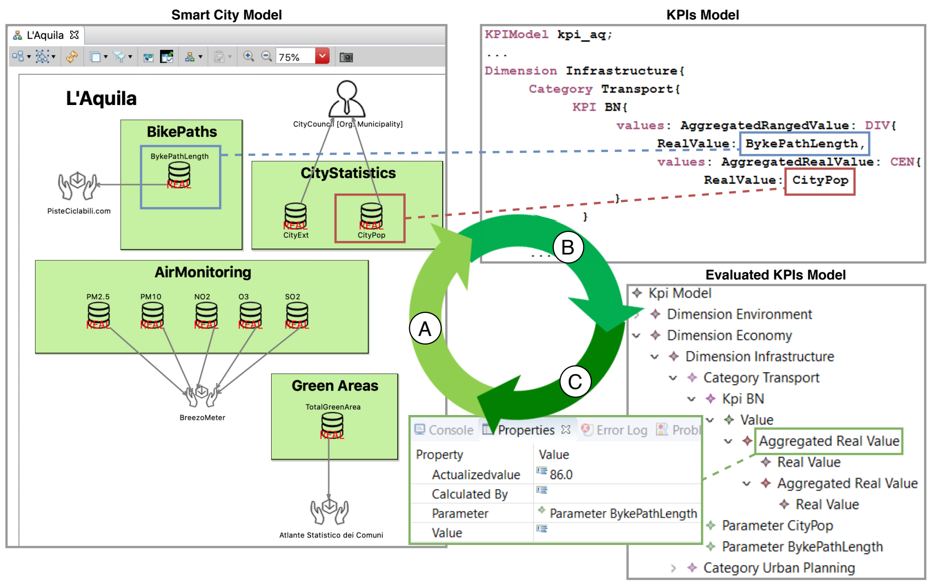
Several architectures have been proposed in the smart cities domain. The minimal requirements that a robust smart cities architecture must meet, such as distributed sensing, integrated management and *flexibility* are given in [8]. In [9], the authors present an approach for designing smart city's ecosystems, by means of a reference architecture called *SmartCityRA*. It represents a way to create smart cities blueprint that can help the instantiation of smart cities projects. They exploit variability modeling and model-driven architecture techniques, to produce a Domain Specific Language (DSL) for modeling smart city systems (i.e., *SmartCityML*). The usability of the approach is further shown through a *Smart Parking* scenario. However, despite this approach provides features for smart cities modeling, like our, it does not support the KPIs modeling and assessment. In [10], a reference architecture for designing a smart city context through the use of Big Data adhering to the NIST (National Institute of Standards and Technology) standard is presented. Here, the focus is more on the design of Big Data processing in *smart environment* contexts. The aim is that of exploiting the proposed conceptual model to create a unified intellectual infrastructure for environmental monitoring. Thus, they do not consider other smart cities contexts, different than environment.

From the perspective of *using data for decision making* in smart cities, different architectures are also provided, as for instance in [11] where data is exploited to support administration processes. Another data-driven approach is presented in [12]. The authors propose a data-driven IoT Software Platform for realizing sustainable *Smart Utilities*, a.k.a. smart services, in order to further develop applications on top of them. Specifically, they provide a service-oriented architecture that makes use of Web standards and protocols. The proposed architecture is scalable over cities of different dimensions and is generalizable to different smart utility domains, other than smart water management. In [13] it is presented a reference architecture to support the development of smart cities platform in order to help stakeholders in making projects, investments and decisions about the cities they manage. In [14] a Distributed-to-Centralized Data Management (D2C-DM) architecture is proposed. It provides different software and services layers, which use distinct data types gathered from physical (e.g., sensors) and non-physical (e.g., simulated data) data sources in the smart city. The aim is to show the easy adaptation of the architecture in contexts with different software requirements.

Nevertheless, despite the availability of multiple architectures supporting smart cities modeling and analysis, either from a wide perspective or focusing on a few dimensions, to the best of our knowledge, none of them specifically targets the modeling and assessment of KPIs that are crucial in the performance evaluation of smart cities.

3 Approach and Proposed Flexible Architecture

In this section we give an overview of the approach for the automatic KPIs assessment in smart cities, which represents the background for this work. Basically the evaluation of a smart city can be summarized in 3 steps: Ⓐ Define the smart city in a way that is processable; Ⓑ Define/select the KPIs of interest; Ⓒ Evaluate the selected KPIs on the subject smart city. In order to support this process, we defined a model-based approach, by identifying smart cities' concepts and the relations among them. Furthermore, we investigated how KPIs can be represented and measured, i.e., what type of calculations and data they require. In MDE, metamodels are central assets that allow designers to formalize application domains and consequently to achieve superior automation [15] in the software life cycle. Indeed, this allowed us to design both a *Smart City metamodel* and a *KPIs metamodel*, on top of which appropriate modeling tools (i.e., graphical and textual concrete syntaxes and editors) can be defined. In particular, the KPIs metamodel reflects the KPIs list for SSCs released by the ITU [4] and conforming to the SDGs [3]. The modeling tools are devoted to smart cities stakeholders, supported by KPIs experts, and allow them to define in a uniform way the smart cities they manage (Ⓐ) together with the KPIs they are interested in (Ⓑ), without knowing technological aspects and abstracting from the target deployment platform. The generated models will be used as input for an *Evaluation Engine* that will interpret and calculate the modeled KPIs for the candidate cities, by giving as output an *evaluated KPIs model* (Ⓒ). We highlight here that the KPIs model can be easily *extended* or *customized* to accommodate specific smart cities requirements as well as the KPIs evolution over time.



```
KPIModel kpi_aq;
...
Dimension Infrastructure{
      Category Transport{
            KPI BN{
                  values: AggregatedRangedValue: DIV{
                        RealValue: BykePathLength,
                        values: AggregatedRealValue: CEN{
                              RealValue: CityPop
                  }
            }
  ...
```

Property	Value
Actualizedvalue	86.0
Calculated By	
Parameter	Parameter BykePathLength
Value	

```
Kpi Model
  Dimension Environment
  Dimension Economy
    Dimension Infrastructure
      Category Transport
        Kpi BN
          Value
            Aggregated Real Value
              Real Value
              Aggregated Real Value
                Real Value
        Parameter CityPop
        Parameter BykePathLength
      Category Urban Planning
```

Fig. 1: Overview of the model-based KPIs assessment approach.

Example. To better understand the approach, we report a basic but realistic example in Fig. 1. We show some simple portions of the models we used to evaluate the smart city of L'Aquila (Italy). We focus on a single KPI, *Bicycle Network* (BN), which is calculated as in equation (1). It measures the length of bicycle paths per 100.000 inhabitants as the ratio between the bicycle paths length and one 100.000th of the city population.

$$BN = \frac{\textit{Bicycle Path Length (km)}}{\frac{1}{100000} \times \textit{City Population}} \tag{1}$$

In the left side of Fig.1 we show a portion of the graphical representation of the model for the city of L'Aquila (Ⓐ), where the concepts needed for the BN KPI calculation have been modeled, e.g., *PisteCiclabili.com* is the provider of the *BikePathLength* data. In the top-right side of Fig.1, instead, there is the portion of the textual representation of the KPIs model (Ⓑ) that refers to the BN KPI and its formula, as in equation (1). Notice that, to be calculated, the BN KPI needs data coming from the smart city model (e.g., *BikePathLength*, *CityPop*). Once these two models have been processed by the evaluation engine, it will give as output the evaluated KPIs model (Ⓒ), on the bottom-right side of Fig. 1, which corresponds to the actualized KPIs model. Note that for those cities with less than 100000 inhabitants, the denominator in equation (1) is evaluated to 1. Thus, for the city of L'Aquila with 69605 inhabitants and 86 km of bicycle paths, the BN KPI has a value of 86 $Km\ per$ 100000 $inhabitants$. What we want to highlight here is that (1) smart governance team can discover *weaknesses* of the assessed cities by interpreting the evaluated KPIs; (2) the approach enables *simulation* and *forecasting* of smart cities performances, by testing different settings in the models.

The Proposed Flexible Architecture. Fig. 2 shows the proposed flexible architecture for KPIs assessment in smart cities supporting the process described above. It is made by six macro-components (also components from here on), representing the main required functionalities, and control- and data-flow between them. The architecture is devoted to several `Stakeholders` that can be divided in two main groups: *(i)* those designing or applying the architecture, i.e., the *Developers Team* (comprising also modelers, DSL engineers and software architects) and the *KPIs Experts*, which are responsible of the design and implementation of the main software components and modeling artifacts; *(ii)* the final users of the software solutions based on the architecture, i.e., *Municipalities*, *Smart Governance Team*, *Ranking Agencies*, *Researchers*, etc. They can be assigned different granting access, i.e., read, write, execute, depending on their profile. For instance, municipalities can be interested in *modeling* the smart cities they manage and *evaluate* KPIs on top of them. On the contrary, ranking agencies might be interested only in the *analysis* and *interpretation* of (open) data about previously evaluated cities.

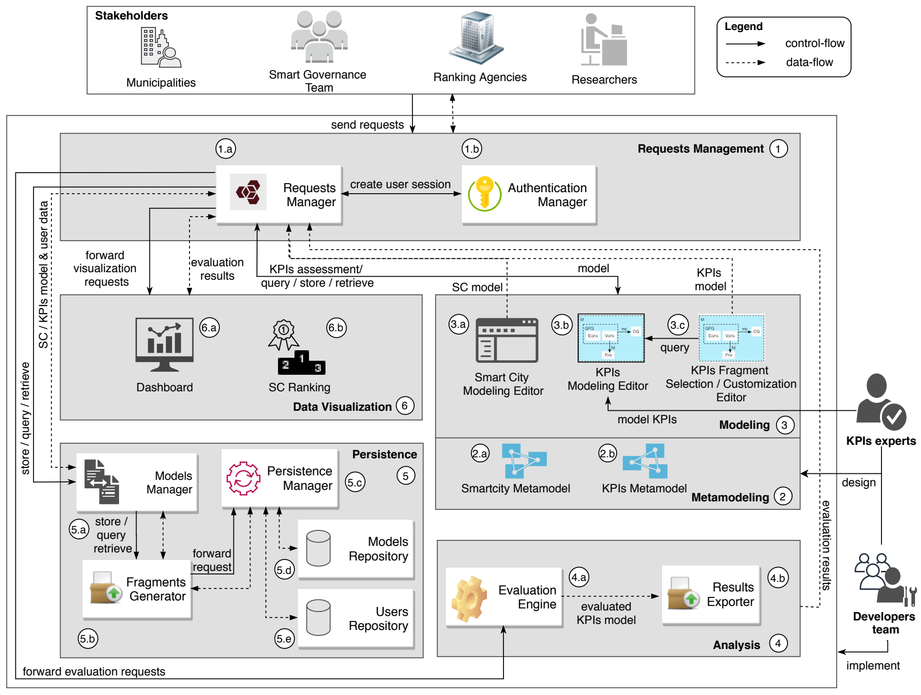


Fig. 2: The Flexible Architecture for the KPIs Assessment in Smart Cities.

The six main components are `Requests Management`, `Modeling`, `Metamodeling`, `Analysis`, `Persistence`, and `Data Visualization`, labeled from **(1)** to **(6)**, respectively. In Fig. 2, solid arrows shape the control-flow among the components and dashed arrows shape the data-flow among them. It is worth noting that data-flow may involve entire models but also raw data may be exchanged, usually persisted via XML or customization of XMI. We now describe each component.

Requests Management. It behaves as the interface among the stakeholders and all the other components of our architecture. Indeed, it handles all the users requests through its two sub-components, the `Requests Manager` **(1.a)** and the `Authentication Manager` **(1.b)**. The `Authentication Manager` handles the users registration, authentication and authorization. It supports the `Requests Manager` every time it needs to *create users sessions* to accomplish the requests they make, based on their access grants. It is an optional component, depending on the deployment style. E.g., it might be not required in standalone specifications of the architecture, while it is suggested for online ones. The `Requests Manager` handles several interactions. It receives and forwards the user's requests to the appropriate components and the requests among components. Three main requests can be handled at this stage, namely (1) the *model* request to the `Modeling` component, from those users (with proper permissions) who needs to model the smart cities they manage and to select/customize the KPIs they are interested in, through the available editors; (2) the *store/query/retrieve* request to the `Models Manager` in the `Persistence` component, from users that want to store/gather models relating to their previously interactions with the system (e.g., municipalities) or from those users with read only permissions who wants to get available open data for their analysis (e.g., researchers); (3) the *forward visualization requests* to the `Data Visualization` component for the graphical visualization and interpretation of the evaluated KPIs. Furthermore, the *model* request to the `Modeling` component can, in turn, give raise to further interactions: during and after the user interplay with the editors, the `Modeling` component can forward to the `Requests Manager` the *query/store/retrieve* request to interact with the `Persistence` component, for querying the `Models Repository` and for storing the produced models as well as the *KPIs assessment* request to be forwarded to the `Evaluation Engine`, via the *forward evaluation requests* interaction.

Metamodeling. It is responsible for hosting the two core metamodels of the model-based approach for the KPIs assessment in smart cities, namely the `Smartcity Metamodel` **(2.a)** and the `KPIs Metamodel` **(2.b)**, and the corresponding tools used to model them. It is also responsible for managing and keep trace of their evolution accomplished cooperatively by the Developers Team and KPIs Experts (*design* relation in Fig. 2), which adapt the two metamodels according to the evolving nature of both smart cities and KPIs. This component is developed at the beginning of the process and should be freezed and stable for enabling the modeling phase. In case evolution scenarios occur, coupled-evolution of the already modeled cities have to be performed in order to be compliant to the new metamodels [16].

Modeling. It is responsible for managing the generated editors required for allowing granted users to both model smart cities and model or select and customize KPIs. Modeling is the activity in which the designer creates / edits the contents of the application. Nowadays multiple modeling tools are available and they differ for various aspects, in which we distinguish textual or diagrammatic concrete syntax, or even more basic modeling editors, like tree view based. For this reason, the usability of the final product is strongly driven by the available modeling tools. Specifically, in this component we find the `Smart City Modeling Editor` **(3.a)** devoted both to experienced and non-expert MDE developers, such as the smart governance team' members that use the system to model the smart city they manage, to subsequently evaluate its performance. Graphical editors are recommended here, for usability reasons, such that to provide user-friendly editors. Differently, the `KPIs Modeling Editor` **(3.b)** is mainly devoted to KPIs Experts who are widely experienced in KPIs and up to date about the evolving KPIs documentation and collection methodology provided at European level. From one hand, they are in charge of modeling KPIs (*model KPIs* in Fig. 2) reflecting the official guidelines. From the other hand, different smart cities managers might be interested in diverse KPIs. For this reason, a certain degree of KPIs selection and customization must be provided. This is accomplished by the `KPIs Fragment Selection/Customization Editor` **(3.c)** that allows users to "*query*" the KPIs model in the `KPIs Modeling Editor` to select and possibly customize given KPIs and further generate the so-called *model fragments*. Borrowing the definition of interesting object structures [17], we define model fragments such KPIs model's internal selection. The main KPIs model includes all the KPIs defined by experts, while model fragments identify the selection made by the user. Eventually, the interaction of stakeholders with the modeling editors brings to the creation of the *Smart City model* (*SC model* from here on) and *KPIs model*, both conforming to the domain models in the `Metamodeling` component, indeed they can be managed by using its generated API.

Analysis. It is responsible for managing the automatic KPIs assessment over smart cities. It includes the `Evaluation Engine` **(4.a)** that is devoted to interpret and calculate the modeled KPIs for one or more candidate smart cities. In particular, it receives KPIs evaluation requests forwarded by the `Requests Manager` (*forward evaluation requests* in Fig. 2) together with the *SC model* and *KPIs model*. After the evaluation, it sends the *evaluated KPIs model* to the `Results Exporter` **(4.b)**, which is in charge of exporting results in different formats (e.g., .csv, .xml or JSON files), depending also on the use that stakeholders intend to make of it (e.g., graphical visualization, textual interpretation, further elaborations) and from the tool which might be used for their visualization and interpretation. Eventually, the `Analysis` component forwards the *evaluation results* to the `Requests Manager` that sends them to the user who submitted the KPIs evaluation request and/or to the `Data Visualization` component.

Persistence. It manages the persistence of all the artifacts involved in the process of KPIs assessment, together with the stakeholders related data, such as their profiles, access grants and authentication data. In particular, it contains five components. The `Models Manager` **(5.a)** acts as the main interface of the `Persistence` component. It receives all the *store/query/retrieve* requests by the `Requests Manager`, together with the accompanying data, such as the *SC/KPIs models & user data*, and it sends the corresponding replies. Moreover, before forwarding the received requests to the `Persistence Manager` **(5.c)**, the `Models Manager` interacts with the `Fragments Generator` **(5.b)** (*store/query/retrieve* in Fig. 2), to handle those cases in which the specific request deals with modeling artifacts and there might be the need of generating model fragments from them. The fragments generation does not apply for those requests addressed to the `Users Repository` and containing only user data, in which case the `Fragments Generator` does not execute any operation and only forward the request to the `Persistence Manager`. Once requests and relative data arrive to the `Persistence Manager`, it is responsible for storing, querying or retrieving data from the appropriate repository, such as `Models Repository` **(5.d)** and `Users Repository` **(5.e)**. In addition, artifacts stored in the `Models Repository` also contain metadata about the users holding their ownership, while the `Users Repository` stores information about users profiles and granting access.

Data Visualization. It is responsible for the *evaluated KPIs model* visualization and interpretation through the `Dashboard` **(6.a)** and `SC Ranking` **(6.b)** components. The former handles the graphical transformation of evaluated KPIs in appropriate and easy to understand charts. The latter shows the ranking among several smart cities whose models are made available and retrieved from the `Models Repository`, only for smart cities comparison purposes. To this aim, the `Data Visualization` receives *forward visualization requests* from the `Requests Manager` with the *evaluation results* from the `Analysis` component.

All the components and their functionalities, except for the `Data Visualization` component, are mandatory to allow architecture specifications to accomplish the task of KPIs assessment in smart cities. On the contrary, to implement its functionalities, a component might not always require all its sub-components, as we will see.

4 Prototypical Implementation

The presented architecture can be specified by using a combination of various technologies and deployment patterns. In this section, we present implementation details about the prototype we developed, corresponding to a *standalone specification*[5], being the architecture entirely implemented as a standalone platform. Eclipse is the release platform we have chosen, which also provides pre-packaged bundles for specific development paradigms. Specifically, the target platform is represented by the *Eclipse Modeling Framework* (EMF)[6] that provides the modeling language to engineer DSLs [18]. The EMF core includes a metamodeling language, called *Ecore*, used for describing domain models, and runtime support for the models including change notification, persistence support with default serialization, and reflective APIs for manipulating objects in the models. On top of the EMF bundle, the `Metamodeling` component **(2)** has been defined and it hosts the two main domain models (**(2.a)** and **(2.b)**) from which the Java code, supporting model manipulation and editors composition, is generated. On top of this layer, two editors are implemented (**(3.a)** and **(3.b)**) to better cope with the composition of the involved models, such as the editing of models with the possibility of filling them with model elements that can be composed. The editors implementation is supported by DSLs that, in general, can be *graphical* or *textual* [19]. From one side, graphical editors provide an intuitive GUI for modelers. On the other side, textual editors provide a support tool to define models as textual specification, which is better transposed by developers. In order to enable the usage of the editors by both experienced and non-expert users (e.g., smart cities stakeholders), the component **(3)** is implemented with two different technologies. The `Smart City Modeling Editor` **(3.a)** is built on top of *Sirius* [20], a graphical concrete syntax generator creating the graphical modeling workbench for modeling smart cities. Smart city projects

[5] Project available at: `https://github.com/gssi/SmartCityModeling.git`
[6] `https://www.eclipse.org/modeling/emf/`

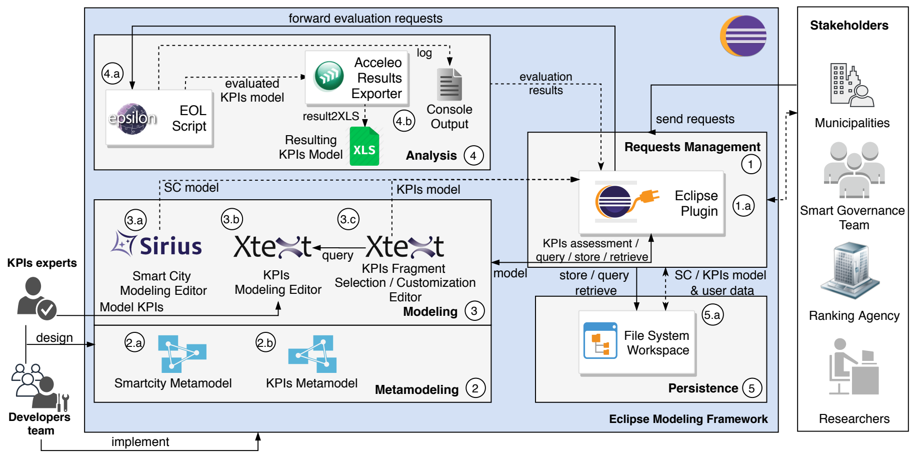


Fig. 3: Standalone specification of our architecture.

are quite large and involve different aspects and view points; since Sirius is based on a view points approach, it suits perfectly in this case. The `KPI Modeling Editor` **(3.b)**, instead, is devised for modelers supporting KPIs experts to specify KPIs and relative calculation formulas. For this reason it has been implemented with a textual concrete syntax by means of *Xtext* [21]. Xtext was chosen because it provides much more “expressiveness” and “agility” to the users, which can edit raw data in a very feasible way. The `KPI Modeling Editor` offers a way to declare how KPIs are calculated in detail, in the perspective of reuse, where the modelers can share their definitions. Lastly, the `KPIs Fragment Selection/Customization Editor` **(3.c)** is implemented by means of a custom reusable mechanism offered by Xtext. In particular, if the KPIs are declared as “reusable operations” in a library, the users can invoke the KPIs needed through this library. When the SC model and KPIs model have been defined, then the `Analysis` component **(4)** can be invoked, thus triggering the KPIs evaluation phase. This process is managed by the `Requests Manager` **(1.a)** that is implemented as an Eclipse plugin organized with the extension point mechanism. It can be activated by file saving operations or even directly by menu entries in the editors. By selecting a SC model and a KPIs model (with specific extensions), a menu entry is enabled and the *EOL script* implementing the `Evaluation Engine` **(4.a)** is triggered. The Epsilon Object Language (EOL) is an imperative programming language part of the Epsilon framework [22] for creating, querying and modifying models built on top of EMF. Basically, the EOL script is a file in the workspace of the project that will be invoked by the plugin. The `Evaluation Engine` generates the evaluated KPIs model and the result will be also printed in the output console of Eclipse. The stakeholders can then request the visualization of the results in two different ways: by inspecting the textual result in the console or by asking to the `Results Exporter` **(4.b)** to generate an .xls file from the evaluated KPIs model. The excel file will be produced by a model-to-text transformation, from the `Results Exporter` implemented in *Acceleo*[7], i.e., one of the most used tools for code generation in MDE.

[7] `https://www.eclipse.org/acceleo/`

5 Evaluation

In this section, we evaluate the *flexibility* of our architecture by giving evidence of (i) alternative *deployment patterns* that can be used, and (ii) the *technology-independent nature* of the architectural components, enabling the use of diverse technologies, also w.r.t. the chosen deployment style. Moreover, we evaluate the *performance* of our architectural approach by running experiments based on the realized prototype.

5.1 Flexibility Evaluation

We now describe two additional specifications of our architecture, namely *hybrid* and *online*. They are currently under development, although they rely on a partial reuse of the implemented standalone specification, thus we focus on the differences w.r.t. it.

Hybrid Specification. In the hybrid specification, depicted in Fig. 4, part of the architectural components are deployed online and part of them reside locally on the user's machine. The Internet layer is in between them. Components **(2)** and **(3)** are the same as in the standalone specification. The `Requests Manager` **(1.a)** presents both a local and a remote instances. The local one is in charge of activating the editors and triggering the remote KPIs evaluation by sending a request, via the internet, to the remote one that will forward the request and the received models to the `Evaluation Engine` **(4.a)**. This step is preceded by an authentication process started by the local `Requests Manager` that will authenticate the local user to the remote `Users Repository` **(5.e)** via the `Authentication Manager` **(1.b)**. For the `Authentication Manager`'s implementation we plan to use the *J2EE framework* with the technologies it exploits for realizing the Model-View-Controller (MVC) architecture (e.g., Spring, Hibernate). The `Users Repository` and the `Models Repository` **(5.d)** are managed by the `Persistence Manager` **(5.c)**. Models can be passed as parameters from the local Eclipse editors, if they are brand new models edited from scratch, or they can be requested to the `Persistence Manager` that will execute the query and retrieve operations on the `Models Repository`. The `Persistence` component **(5)** has different tasks that range over the usual repository operations, to the query management, in order to extract fragments of the models through the `Fragment Generator` **(5.b)**, which can be realized by means of a DSL similar to that used for the `KPIs Fragment Selection/Customization Editor` **(3.c)** allowing for querying models. Repositories operations are delegated to a repository manager called *MDEForge* [23] implementing the `Persistence Manager`. It consists of a set of core services that permit to store and manage typical modeling artifacts and tools, specifically conceived for models. It comes with APIs that can be used to interact with the repository functions without using the provided web interface. This allows MDEForge to be easily integrated in the infrastructure. Lastly, the `Models Manager` **(5.a)** will be interposed between the MDEForge and the `Requests Manager`, as an interface of the `Persistence` component. When the required models are available for the analysis phase, the `Evaluation Engine` **(4.a)** implemented with a Java model parser can be invoked. Through model interpretation also called compilation, the input

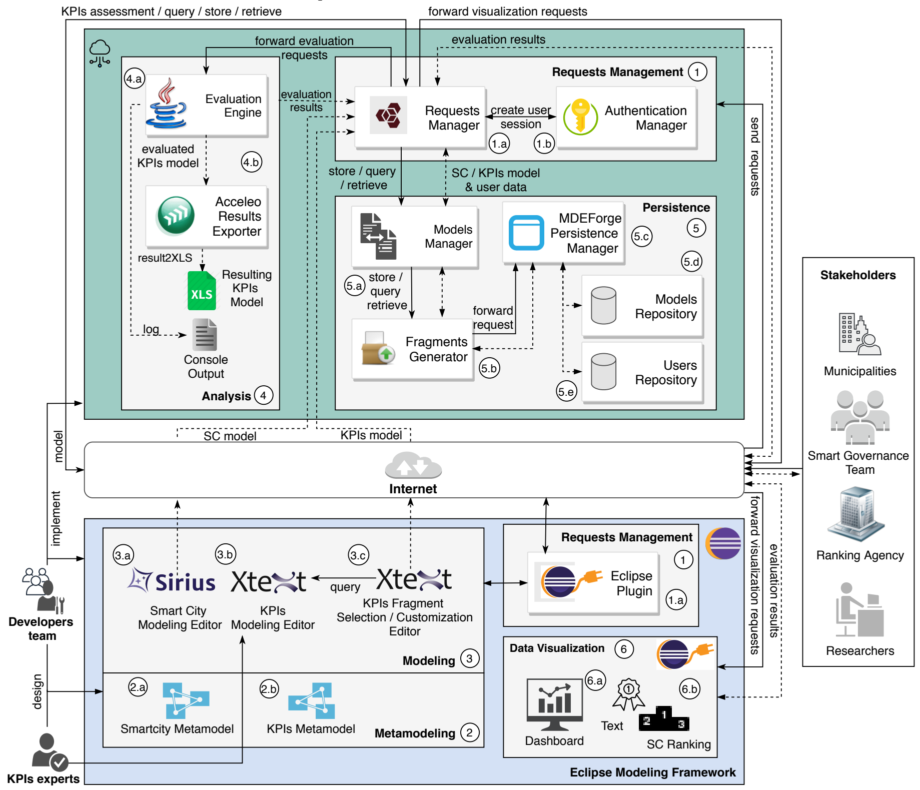


Fig. 4: Hybrid specification of our architecture.

models are directly used to run the system [24] or to invoke other actions during the interpretation at runtime. As before, the analysis phase will generate the evaluated KPIs model, however the console output in the hybrid specification is only used for testing purpose, being accessible only from the server-side. In this specification, the Acceleo `Results Exporter` can export results in other format than .xls file or it can expose itself an API for results visualization purposes, to be forwarded by the `Requests Manager`. We then plan to provide a local `Data Visualization` component **(6)** as an Eclipse plugin. It can read the evaluation results both by receiving the generated results file or by listening to the dedicated API, thus implying a sort of asynchronous call to the `Evaluation Engine`. The `Data Visualization` populates the charts shaping the results of the KPIs evaluation in an Eclipse view used by the stakeholders.

Online Specification. The online specification has the peculiarity of being completely deployed online. For lack of space, we do not show its deployment design[8] and we describe only its relevant differences w.r.t. the hybrid specification. Being everything online, the main instrument to use the platform is the browser, that provides access to a web application including the different components. More specifically, the `Modeling`

[8] For the interested readers it can be found at `https://bit.ly/3bqbqG2`

component **(3)** will be implemented through *Eclipse Theia*[9] allowing to run Eclipse online, with all the benefits of having an in-browser extensible IDE. For the editors, the two candidate technologies allowing for running modeling editors in web browsers are *Eugenia Live* [25] for the `Smart City Modeling Editor` **(3.a)** and Xtext (from version 2.9 on) for the `KPIs Modeling Editor` **(3.b)**.

The advantage of an online environment supports one of the main problems that slows down the path of MDE towards a standard: the reluctance in installing different tools, most of them academic, with all the related issues linked to safety and reliability. These aspects cannot be neglected in an industrial scenario. Moreover, collaborative repositories have been extensively proposed and investigated in MDE [26], highlighting multiple challenges. Among these, visibility of the repositories stored artifacts seem to be one of the hot topic in industry. Multiple resolutions have been proposed to tone down the reluctance in employing these tools, and extended visibility management seems to be needed [27] to assure that only artifacts intentionally shared will be visible to the community and not the models including intellectual property rights.

Alternative Candidate Technologies. We did an analysis about available languages and technologies and their suitability for implementing our architecture. Several valid alternatives exist to implement the different components of Fig. 2, as shown in Table 1. This list is not intended to be exhaustive, given the panorama of possible integrations, but we give an idea of available tools. In the second column of the table, ✓denotes that the corresponding component is mandatory, independently of the used deployment style, while $\approx$ denotes that the component may be missing, e.g., because its offered functionality may be omitted without compromising the functioning of the approach (e.g., the `Data Visualization`) or because it is not required for the used deployment (e.g., the `Authentication Manager` in a standalone instance).

5.2 Performance Evaluation

To evaluate the performance of our architecture, we performed a set of experiments by running the standalone specification, using a 6 core CPU running at 2.2GHz, with 16Gb memory. The goal of our experiments is that of checking the evaluation engine execution time w.r.t. the *size of the input models*, i.e., the number of elements in the models. Thus, we designed a smart city model, in which we instantiated every concept of the metamodel, and a KPIs model. In particular, according to the ITU, KPIs are hierarchically organized [4]. A KPIs model is composed by $1 \ldots n$ dimensions, also containing sub-dimensions, each composed by $1 \ldots n$ categories that, in turn, contain $1 \ldots n$ KPIs. The used KPIs model is initially made by one dimension with one category of 8 KPIs, thus to cover all the calculations defined in the KPIs Metamodel.

[9] `https://theia-ide.org` [10] `https://bit.ly/3cGvJAC`
[11] `https://bit.ly/2z6pF5A` [12] `https://www.eclipse.org/atl/`
[13] `https://bit.ly/2WZcYl6` [14] `https://bit.ly/3eQQef6`
[15] `https://bit.ly/2Z9bzuR` [16] `https://github.com/neo4emf/Neo4EMF`
[17] `https://emfjson.github.io/` [18] `https://www.eclipse.org/emfstore/`
[19] `https://www.mysql.com/` [20] `https://bit.ly/353KD0P`
[21] `https://spring.io/` [22] `https://www.eclipse.org/xtend/`

Table 1: Architecture flexibility in terms of required components and technologies.

Components	Mandatory	Candidate Technologies
Requests Manager	✓	Eclipse plugin, Java
Authentication Manager	≈	Java
Smartcity Metamodel	✓	Ecore, Kermeta [28], UML
KPIs Metamodel	✓	Ecore, Kermeta, UML
Smart City Modeling Editor	✓	Sirius, Eugenia Live, Eugenia
KPIs Modeling Editor	✓	Xtext, EMFText [29]
KPIs Fragment Editor	✓	Xtext, OCL[10], EMF-Fragments[11]
Evaluation Engine	✓	Java, EOL Script, ATL[12], ETL[13]
Results Exporter	≈	Acceleo, JET[14], EGL[15], Xtend[22]
Models Manager	≈	Java, MDEForge
Fragments Generator	≈	EMF + Java + OCL
Persistence Manager	✓	MDEForge, Neo4EMF[16], Relational DB, EMFJson[17]
Models Repository	✓	MDEForge, EMFStore[18]
Users Repository	≈	NoSql [30], Mysql[19], MSSql[20]
Dashboard	≈	Spring[21], other J2EE or JS-based frameworks
SC Ranking	≈	Spring, other J2EE or JS-based frameworks

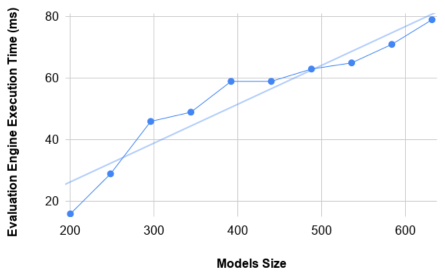


Fig. 5: $Exp1$: increasing the number of evaluated smart cities.

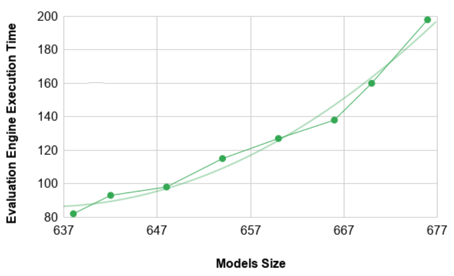


Fig. 6: $Exp2$: increasing the complexity in the calculation of each modeled KPI.

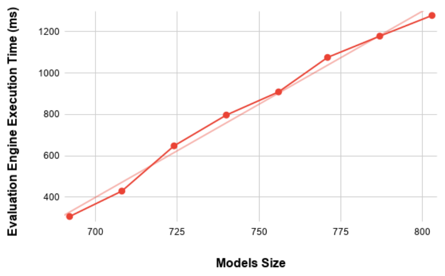


Fig. 7: $Exp3$: make each KPI of type *range*.

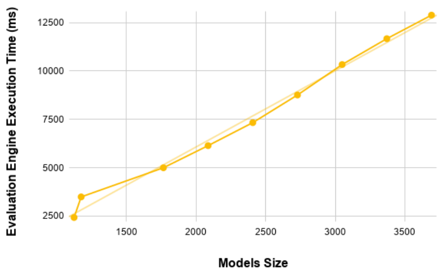


Fig. 8: $Exp4$: increasing the number of KPIs.

In Fig. 5, we show the results of our first experiment $Exp1$. For each execution run of the evaluation engine, we increment the number of modeled smart cities in the SC model from 1 to 10 and we measure the 8 KPIs in the KPIs model for each of them. As shown in Fig. 5, the models size goes from 200 to 632 elements and the execution time goes from 16 milliseconds (ms) to 79 ms. Fig. 6 reports the results of $Exp2$ for which we started by giving as input to the evaluation engine the SC model made by 10 smart cities, which remain fixed, and the KPIs model as in $Exp1$. Then, at every run we increase the complexity of the KPIs calculations, by adding new nested operations, one KPI at a time. This impacts on the time required to measure each KPIs. In Fig. 6 we can observe that the size of the models goes from 638 to 676 and the execution time goes from 82 ms to 198 ms. Interestingly, despite the models size does not increase significantly, the execution time particularly increases in the last run. This is due to the fact that this run involves a KPI whose calculation combines the *range* operation, i.e., the most time consuming one, with a basic operation (e.g., AVG). From this observation, we designed $Exp3$ (Fig. 7) such that, at every run, we add a *range* calculation in the definition of each KPI, one KPI at a time. In Fig. 7, we can observe that the size of models slightly increases from 692 to 803 elements (only 127 elements more than the last run in $Exp2$) while the execution time ranges from 306 ms to 1279 ms, by showing a considerable increase, thus confirming our prediction about the time consuming of calculations including the range operation. However, the overall execution time is still reasonable for the given models size. Eventually, in $Exp4$ (Fig. 8) at every run we increment by one the number of dimensions in the KPIs model, where each dimension includes 8 KPIs with complex calculations. Consequently, the number of evaluated KPIs goes from 16 in the first run to 80 in the last one, always assessed on top of the 10 smart cities in the SC model. This means that in the last execution we assessed 800 KPIs in the same run. Fig. 8 depicts that the size of the models goes from 1124 to 3692 elements and the execution time ranges from 2426 ms to 12892 ms.

Summarizing, these experiments point out two main findings: (i) the efficiency in terms of the evaluation engine's execution time, since all the experiments show a linear or polynomial (of degree 2) increase of the execution time w.r.t. the increasing models size; (ii) promising scalability results showed by $Exp4$, indicating that the system takes 12.9 seconds for assessing 800 KPIs over 10 smart cities.

Threats to validity. The settings of the input parameters in the evaluation might internally bias our experimentation. Both the SC model and the KPIs model lead to the execution of calculations of diverse complexity, depending on the size of the two models and the number of KPIs, but the overall evaluation procedure is not affected. For these reasons, we considered incrementally complex models in each run, to trigger more complex calculation and smooth biases in the output results. As external threats to validity, we are aware that we need to perform the KPIs evaluation by using SC models of real smart cities, but we leave this point as part of our future work, where we plan to evaluate the system by involving real stakeholders, also to evaluate the provided editors.

6 Conclusion and Future Work

In this paper we presented an architecture supporting a model-based approach for the KPIs assessment in smart cities. Its goal is to provide a robust and flexible platform for the performance evaluation of smart cities, to be easily used by smart cities stakeholders, during the decision making and planning process.

As future work we are finalizing the implementation of the *hybrid* and *online* specifications and designing an experiment to evaluate the *feasibility* and *usability* of the methodology with the involvement of real smart cities stakeholders who have to make use of the provided modeling and analysis tools and data visualization facilities. Finally, we consider the integration with legacy data formats such as CityGML.

Acknowledgment

This work was partially supported by the Centre for Urban Informatics and Modelling, National Project, GSSI as well as by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development.

References

1. D. Mutiara, S. Yuniarti, and B. Pratama, “Smart governance for smart city,” *IOP Conf. Series: Earth and Environmental Science*, vol. 126, pp. 012–073, 2018.
2. Directorate-General for Environment (European Commission), Intrasoft International, University of the West of England (UWE). Science Communication Unit, “Indicators for sustainable cities,” April 2018.
3. European Commission, “Europe 2020 A European strategy for smart, sustainable and inclusive growth,” March 2010.
4. International Telecommunication Union (ITU), “Collection Methodology for Key Performance Indicators for Smart Sustainable Cities,” 2017, https://bit.ly/2SkSZfi.
5. E. Ferro, B. Caroleo, M. Leo, M. Osella, and E. Pautasso, “The Role of ICT in Smart City Governance,” in *Int. Conf. for e-Democracy and Open Government*, 2013.
6. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, Second Edition*. Morgan & Claypool Publishers, 2017.
7. P. Mohagheghi and J. Aagedal, “Evaluating quality in model-driven engineering,” in *Int. Workshop on Modeling in Software Engineering*. IEEE, 2007, pp. 6–6.
8. 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 *28th Annual ACM Symposium on Applied Computing (SAC)*. ACM, 2013, p. 1722–1727.
9. M. Abu-Matar and R. Mizouni, “Variability modeling for smart city reference architectures,” in *IEEE Int. Smart Cities Conf.*, 2018, pp. 1–8.
10. 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.
11. R. Wenge, X. Zhang, C. Dave, L. Chao, and S. Hao, “Smart city architecture: A technology guide for implementation and design challenges,” *China Communications*, vol. 11, no. 3, pp. 56–69, 2014.
12. Y. Simmhan, P. Ravindra, S. Chaturvedi, M. Hegde, and R. Ballamajalu, “Towards a data-driven iot software architecture for smart city utilities,” *Softw. Pract. Exp.*, vol. 48, no. 7, pp. 1390–1416, 2018.
13. E. F. Z. Santana, A. P. Chaves, M. A. Gerosa, F. Kon, and D. S. Milojicic, “Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture,” *ACM Comput. Surv.*, vol. 50, no. 6, 2017.
14. A. Sinaeepourfard, S. A. Petersen, and D. Ahlers, “D2C-SM: Designing a Distributed-to-Centralized Software Management Architecture for Smart Cities,” in *Digital Transformation for a Sustainable Society in the 21st Century*. Springer, 2019, pp. 329–341.

15. L. Bettini, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Quality-driven detection and resolution of metamodel smells," *IEEE Access*, vol. 7, pp. 16 364–16 376, 2019.
16. D. Di Ruscio, L. Iovino, and A. Pierantonio, "What is needed for managing co-evolution in mde?" in *Int. Workshop on Model Comparison in Practice*. ACM, 2011, p. 30–38.
17. E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. L. Traon, "Metamodel-based test generation for model transformations: an algorithm and a tool," in *Int. Symp. on Software Reliability Engineering*, 2006, pp. 85–94.
18. D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. Polack, "Requirements for domain-specific languages," in *Workshop on Domain-Specific Program Development*, 2006.
19. P. Veisi and E. Stroulia, "AHL: Model-Driven Engineering of Android Applications with BLE Peripherals," in *Int. Conf. on E-Technologies*. Springer, 2017, pp. 56–74.
20. V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of dsm graphical editor," in *Int. Conf. on Intelligent Engineering Systems (INES)*, 2014, pp. 233–238.
21. L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing, 2016.
22. D. S. Kolovos, R. F. Paige, and F. A. Polack, "The Epsilon Object Language (EOL)," in *European Conf. on Model Driven Architecture-Foundations and Applications*. Springer, 2006, pp. 128–142.
23. F. Basciani, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, "MDEForge: an Extensible Web-Based Modeling Platform," in *CloudMDE@MoDELS*, 2014, pp. 66–75.
24. S. J. Mellor and M. Balcer, *Executable UML: A foundation for model-driven architectures*. Addison-Wesley, 2002.
25. L. M. Rose, D. S. Kolovos, and R. F. Paige, "Eugenia live: a flexible graphical modelling tool," in *Extreme Modeling Workshop*, 2012, pp. 15–20.
26. J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Collaborative repositories in model-driven engineering," *IEEE Software*, vol. 32, pp. 28–34, 2015.
27. F. Basciani, J. D. Rocco, D. D. Ruscio, L. Iovino, and A. Pierantonio, "Model repositories: Will they become reality?" in *CloudMDE@MoDELS*, 2015.
28. J.-M. Jézéquel, O. Barais, and F. Fleurey, "Model driven language engineering with kermeta," in *Generative and Transformational Techniques in Soft. Eng. IV: Int. Summer School*. Springer, 2009, pp. 201–221.
29. F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, "Model-based language engineering with emftext," in *Generative and Transformational Techniques in Soft. Eng. IV: Int. Summer School*. Springer, 2013, pp. 322–345.
30. C. Strauch, U.-L. S. Sites, and W. Kriha, "NoSQL databases," *Lecture Notes, Stuttgart Media University*, vol. 20, p. 24, 2011.