PeerJ Computer Science

Modeling and management of usage-aware distributed datasets for global Smart City Application Ecosystems

Johannes M. Schleicher[1], Michael Vögler[1], Christian Inzinger[2] and Schahram Dustdar[1]

[1] Distributed Systems Group, TU Wien, Vienna, Austria
[2] Software Evolution & Architecture Lab, University of Zürich, Zürich, Switzerland

ABSTRACT

The ever-growing amount of data produced by and in today's smart cities offers significant potential for novel applications created by city stakeholders as well as third parties. Current smart city application models mostly assume that data is exclusively managed by and bound to its original application and location. We argue that smart city data must not be constrained to such data silos so that future smart city applications can seamlessly access and integrate data from multiple sources across multiple cities. In this paper, we present a methodology and toolset to model available smart city data sources and enable efficient, distributed data access in smart city environments. We introduce a modeling abstraction to describe the structure and relevant properties, such as security and compliance constraints, of smart city data sources along with independently accessible subsets in a technology-agnostic way. Based on this abstraction, we present a middleware toolset for efficient and seamless data access through autonomous relocation of relevant subsets of available data sources to improve Quality of Service for smart city applications based on a configurable mechanism. We evaluate our approach using a case study in the context of a distributed city infrastructure decision support system and show that selective relocation of data subsets can significantly reduce application response times.

Submitted 14 January 2017
Accepted 19 April 2017
Published 22 May 2017

Corresponding author
Johannes M. Schleicher,
schleicher@dsg.tuwien.ac.at

Academic editor
Chee Shin Yeo

Additional Information and Declarations can be found on page 22

DOI **10.7717/peerj-cs.115**

Subjects Distributed and Parallel Computing, Software Engineering
Keywords Smart city application engineering, Data management, Data migration, Quality of service

INTRODUCTION

Sparked by the rapid adoption of the smart city paradigm and fueled by the rise of the Internet of Things, today's metropolises have become data behemoths. With every day that passes more and more areas of cities around the globe start accumulating and producing data. These areas cover building management, traffic and mobility systems, energy grids, water and pollution management, governance, social media, and many more. This plethora of heterogenous data about various aspects of a city represents a vital foundation for decision and planning processes in smart cities. The advent of more and more open data initiatives around the globe, covering cities like London (https://data.london.gov.uk/), Vienna (https://open.wien.gv.at/site/open-data/), New York (https://nycopendata.socrata.com/), and many more underlines the importance of opening up data to the public to inspire and support novel applications. Even though these initiatives are gaining momentum,

they still only cover a fraction of the available data of a city, missing many vital sources, especially when it comes to more sensitive areas like building management, energy grids, or public transport guidance systems. Currently, this data is mostly isolated and restricted to certain application areas, data centers, organizations, or only accessible in a specific city. This isolation creates data silos, which lead to transitive restrictions that apply to the models and applications that build upon them, confining them to their initial application domains. Today's smart cities however, represent heterogeneous, dynamic, and complex environments that rely on emerging interactions in order to operate effectively. These interactions are an essential element of Smart City Applications (*Schleicher et al., 2016a*) and not only important in an intracity context, but also a key element to enable the future Internet of Cities (*Schleicher et al., 2015b*), an interconnected system of systems that spans multiple cities around the globe. To pave the way for such applications we need to break up the traditional notion of data silos to enable ubiquitous access to the valuable data they contain. In the context of smart cities, an approach is required that respects the complexities of this domain, specifically the need to effectively describe a large variety of heterogenous data sources along with relevant subsets. Additionally, it has to be able to capture important data set characteristics (e.g., size, update frequency, costs), respect essential security and compliance constraints, as well as ensure efficient and seamless data access.

In this paper, we present **S**mart **D**istributed **D**atasets (SDD), a methodology and framework to enable transparent and efficient distributed data access for data sources in smart city environments. We introduce a system model that provides a simple abstraction for the technology-agnostic description of data sources and their subsets with the ability to express varying data granularities and specific characteristics common in the smart city domain. Based on this abstraction, we present the SDD framework, a middleware toolset that enables efficient and seamless data access for smart city applications by autonomously relocating relevant subsets of available data sources to improve Quality of Service (QoS) based on a configurable mechanism that considers request latency, as well as costs for data transfer, storage, and updates. We provide a proof of concept implementation of the SDD framework and evaluate it using a case study in the context of a distributed city infrastructure decision support system. For this case, we show that selective relocation of data subsets using the SDD framework can significantly improve QoS by reducing response times by 66% on average.

The remainder of this paper is structured as follows. In 'Motivation' we present a motivating scenario and identify the associated key requirements. We introduce the system model underlying SDD in 'System Model' and present the SDD framework along with a detailed discussion of its components in 'The SDD Framework'. In 'Evaluation' we evaluate our approach using a case study from the smart city domain. Related work is discussed in 'Related Work', followed by a conclusion and outlook on future research in 'Conclusion'.

MOTIVATION

In this paper, we base our discussion on our recent smart city research within URBEM (http://urbem.tuwien.ac.at), a research initiative of the city of Vienna and TU Wien. Within

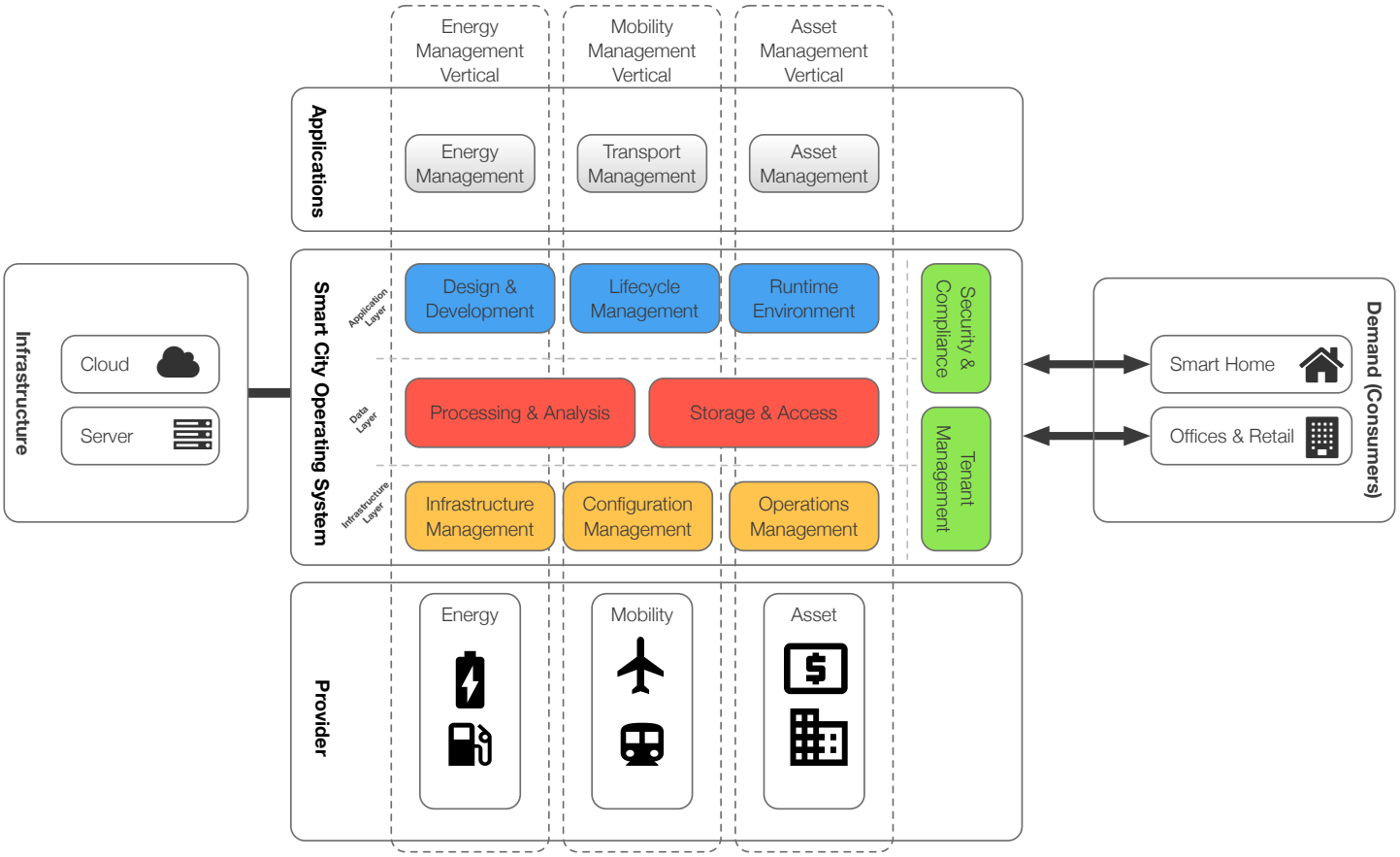

Figure 1 **The Smart City Application Ecosystem with the Smart City Operating System at its core.**

URBEM, we proposed the Smart City Application Ecosystem (SCALE) (*Schleicher et al., 2016a*) shown in Fig. 1 as a streamlined way for modeling, engineering, and operating future smart city applications based on a common abstraction, the Smart City Operating System (SOS) (*Vögler et al., 2016*). The aim of SCALE is to enable stakeholders, citizens, and practitioners in a smart city environment to build novel kinds of applications that can utilize the newfound capabilities emerging through the IoT, as well as accessing the massive amounts of data emitted by the city in an efficient way. Using SCALE, we created the URBEM Smart City Application (USCA) (*Schleicher et al., 2016c*), a holistic, interdisciplinary decision support system for the city of Vienna and a number of key stakeholders. We argue that such applications will evolve to become composable, interchangeable abstractions of capabilities similar to the applications known from today's smart phones, but on a much larger scale. This evolution in turn is an essential step towards the so-called Internet of Cities (*Schleicher et al., 2015b*), an open and dynamic market place where applications can seamlessly interact and be exchanged between cities around the globe.

To enable these applications as well as this vital open exchange it is essential to provide means to expose and access data in an efficient, secure, and predictable way. Currently,

most of the data in a smart city context is confined to certain application areas and stakeholder data centers within a specific city. Open data initiatives around the globe, while crucial, still only expose a certain fraction of the available data, missing out on many important domains, especially when data is stored in legacy systems without openly accessible interfaces or underlies strict security and compliance constraints. This data lies dormant beyond its initial use case even though it could provide essential input for a wide range of smart city applications. The ability to benefit from incorporating new data sources as they evolve, for example to enhance decision support and planning, or to be applied to new cities or novel domains, is hindered by the inability of these applications to access the necessary data. Developers of smart city applications, however, need to be able to utilize and integrate as much relevant data as possible to generate maximum user benefit as well as applicability in as many cities as possible. Stakeholders on the other hand, as willing as they might be to expose this data, are mostly bound by the complex constraints of their specific environment. The dynamic, emergent nature of interactions in and between smart city applications means they are not a priori aware that their data sources might become valuable assets if made accessible. This leads to a problematic stalemate between practitioners and stakeholders in the smart city domain, hindering essential innovation and application.

To overcome this impasse, a mechanism is required that enables flexible, stable, and efficient data access, while providing a simple and tailored way to make data sources available, which still respects security and compliance constraints. Specifically, we identify the following requirements in the context of our domain:

- The ability to describe data sources using an evolvable and technology-agnostic abstraction.
- The ability to describe subsets of these data sources along with relevant characteristics in the context of security, compliance and costs (e.g., effort to generate, store, query, and update particular subsets or the underlying data source as a whole).
- An efficient way to access this data in a transparent way, independent of geographic location while still improving QoS.

SYSTEM MODEL

In order to address the previously outlined requirements, we need an abstraction to model and describe the relevant data entities in our domain. As foundation for said abstraction we use MADCAT (*Inzinger et al., 2014*) and its extensions, which we introduced in Smart Fabric (*Schleicher et al., 2015a*). We presented an infrastructure agnostic deployment model with the following abstract concepts: *Technical Units (TUs)* to describe applications as well as application components, *Infrastructure Specifications (IS)* describe infrastructure resources, *Deployment Units (DUs)* to describe how to deploy and *TU* on an *IS* and an *Deployment Instance (DI)* represented such an actual deployment. In this paper, we extend this model with the ability to describe and incorporate data entities from the smart city domain. Specifically, we introduce the additional concepts of *Data Units (DAUs)* to model data sources as well as *Data Instances (DAIs)* to describe specific deployments of *DAUs*

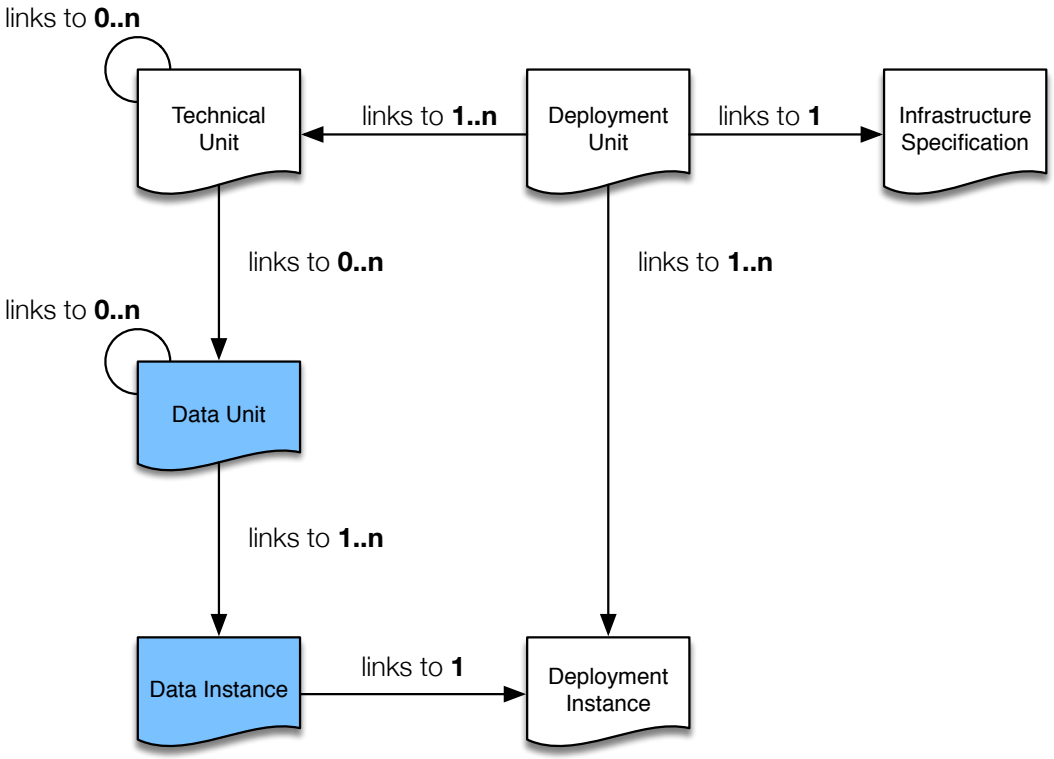

Figure 2 **Relations between Technical Unit, Deployment Unit, Infrastructure Specification and Deployment Instance including the newly introduced Data Unit and Data Instance.**

on certain *DIs*, along with the ability to link *TU* to *DAUs*. Additionally, we provide an implementation of the abstract concepts along with the proposed methodology extensions for the SDD Framework. We again choose JSON-LD (http://json-ld.org/) as data format for our concept description as a simple, both human and machine readable, pragmatic, and extensible representation that also allows us to interlink the relevant concepts with each other. Figure 2 shows an overview of all concepts, including the relations of the newly introduced *DAU* and *DAI* (shown in blue) to the previously existing concepts *Deployment Instance (DI)* and *Technical Unit (TU)*. In the following, we discuss the introduced concepts in more detail.

Data Unit

DAUs describe data sources including its subsets. Listing 1 shows an example of such a *DAU* for a buildings data source in the URBEM domain. As JSON-LD document, a *DAU* can start with a `context` to set up common namespaces. This is followed by the `type` attribute to identify the corresponding kind of abstraction in our model space. The next attribute is the `name` as *URN* to identify the unit, along with a `version` attribute enabling versioning and therefore evolvable *DAUs*. The `creationDate` and the `lastUpdate` define when the unit description was initially created respectively last updated. The next section is `metainformation` about the described data source. It contains a `type` attribute that defines the type of the data source, types can be REST, SOAP, any database, streaming data

or file based. In our example listing it is used to describe a document oriented MongoDB (https://www.mongodb.com/) based data source. The `schema` attribute can link to a corresponding schema, which based on the `type` and can be anything of the likes of a SQL schema, JSON schema, WADL (https://www.w3.org/Submission/wadl/) or WSDL (https://www.w3.org/TR/wsdl). The next attribute is `securityConstraints` and in the context of this section it is used to define who is allowed to access the unit file. A security constraint can be a link to an OAuth (https://oauth.net/) authority, LDAP distinguished name (DN) or any other corresponding authentication and authorization scheme. This is a vital element to ensure the compliance and security constraints of this domain can be met on any level of detail and is again used when describing specific facets of a data source. The last element in the metainformation section is the `dataUnits` attribute, which allows to link a *DAU* to other corresponding *DAUs* enabling the description of linked data sources. The next section `views` allows to express multiple facets of the data source enabling a fined grained level of control about its aspects. Each view has a `name` attribute for identification as well a a `link` to express how to access it. In case of the our example this is a *URL* to the corresponding rest resource. The next section within a view is the *updateFrequency*. It allows to express how often a view is being updated (`period,times`), how long such an update takes (`updateTime`) as well as how much of the resource this update represents (`fraction`). The `size` attribute gives an indication of the expected size of the view. This is followed by a `securityConstraints` attribute that again can be a set of links to a corresponding authentication or authorization scheme supporting the previously mentioned methods. In this section it is used to express security and compliance constraints for specific fractions of a data source, which allows for a fine very grained level of control. The last attribute in a view is the *dataInstances* attribute, which links the view and its corresponding *DAU* to one or more *Data Instance (DAI)* by referencing their names.

Listing 1: Data Unit—Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DataUnit",
  "name": "urn:buildings:vienna",
  "version": "1.0"
  "creationDate": " 2017-01-07 13:04:03 +0100 ",
  "lastUpdate": " 2017-01-07 13:04:03 +0100"

  "metainformation": {
    "type":"nosql:mongodb",
    "schema": "http://sdd.dsg.tuwien.ac.at/urbem/vienna/buildings",
    "securityConstraints":[{
      "type":"ldap"
      "url":"10.2.0.112"
      "dn":"CN=buildings-vienna",
      ...
      }],
    "dataUnits":[...]
  }

  "views":[{
    "name":"urn:buildings:vienna:buildingblocks",
```

```
    "link":"/buildingblocks/",
    "updateFrequency":{
      "period":"yearly",
      "times": "1",
      "fraction": "10"
      "updateTime" : "2300"
      },
    "size":"10000303",
    "securityConstraints":[...]
    "dataInstances":[...]

    },
    "name":"urn:buildings:vienna:buildings",
    "link":"/buildings/",
    ...]
}
```

Data Instance

A *DAI* represents a specific deployment of a *DAU* on a *DI*. There can be multiple *DAIs* for a *DAU* representing different deployed views, where a specific *DAI* contains a subset of the views specified in the *DAU*, i.e., $DAI \in \mathcal{P}(DAU)$. A *DAI* specifies `context`, `type`, `name` and `version`, as well as a `dataUnit` to reference the corresponding *DAU*. This if followed by `creationDate` and `updateDate` to define when the instance was created as well as last updated. The next attribute is `deploymentInstance`, which contains a *DI* name and is used to link the *DAI* to a corresponding *DI*. Finally, the `metainformation` element allows to store additional information about the specific data instance in an open key value format that can later be used by the framework components to support transfer decisions, examples would be `accessFrequency` of this specific *DAI* or other non functional characteristics.

Listing 2: Data Instance—Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DataInstance",
  "dataUnit":"urn:buildings:vienna"
  "name": "urn:buildings:vienna:buildingblocks",
  "version":"1.0",
  "creationDate":"2017-01-07 13:04:03 +0100"
  "lastUpdate":"2017-01-07 13:04:03 +0100"
  "deploymentInstance":"CitizenInformationSystem/DedicatedServer"
  "metainformation":{...}
}
```

For further information regarding the elements of a *TU*, *DU*, *DI*, *IS* as well as *DAU* and *DAI*, we provide detailed example representations of all of them in the corresponding Bitbucket repository (https://bitbucket.org/jomis/smartdata).

THE SDD FRAMEWORK

In this section, we introduce the SDD framework for enabling usage-aware distributed datasets, to address the previously introduced requirements. We begin with a framework overview, followed by a detailed description of all framework components and conclude with a comprehensive description of our proof of concept implementation.

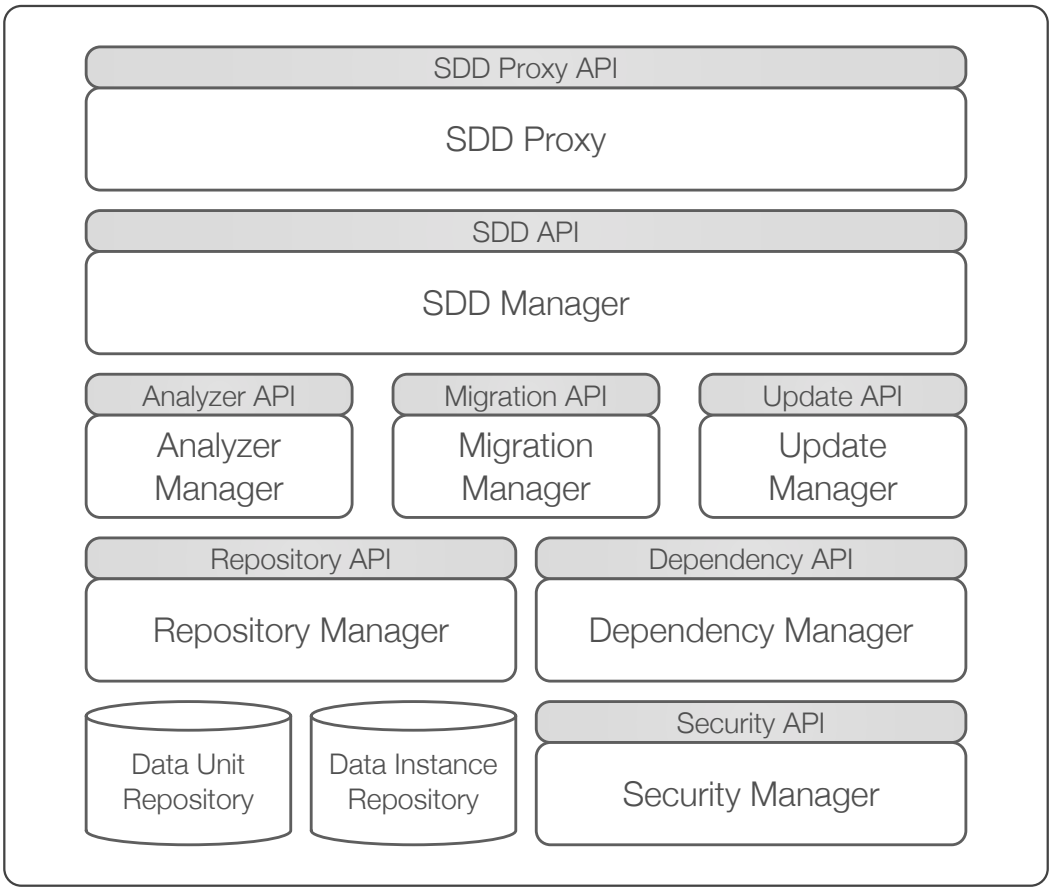

Figure 3 **SDD framework overview.**

Framework rationales

The Framework with an overview of its main components shown in Fig. 3 follows the microservice (*Newman, 2015*) architecture paradigm. It consists of eight main components where each of these components represents a microservice. The components utilize both service based as well as message-oriented communication to exchange information. Specifically, we distinguish three different queue types: an *Analyzer Queue*, a *Handler Queue* as well as an *Update Queue*, which will be explained in more detail in the context of the corresponding components.

Additionally, the framework utilizes the principle of *Confidence Elasticity* a concept we introduced and successfully applied in our previous work (*Schleicher et al. 2016b*; *Schleicher et al. 2015a*). In this framework we use the concept in the *Update Manager* and *Migration Manager* component to select a suitable *Migration* respectively *Update Strategy* for a specific data source. Each strategy is associated with a confidence value ($c \in \mathbb{R}, 0 \leq c \leq 1$), with 0 representing no certainty and 1 representing absolute certainty about the produced result. This convention allows the framework to configure certain confidence intervals to augment the process of choosing an applicable strategy within these two components. These confidence intervals are provided as configuration elements for the framework. If

the confidence thresholds are not met, the framework follows an escalation model to find the next strategy that is able to provide results with higher confidence until it reaches the point where human interaction is necessary to produce a satisfactory result. In the context of the *Migration and Update Manager* components, this means if an migration or update of a data source cannot be performed with a satisfactory confidence the escalation model will prompt for human interaction to perform said migration or update.

SDD Proxy

The *SDD Proxy* acts as a transparent Proxy between clients and data sources (*DAIs*). The proxy itself has two main responsibilities. First, it submits all incoming requests to a *Analyzer Message Queue* before forwarding them to the requested data source. Second, it listens to the *Handler Message Queue* for potential redirections to be taken for a specific request. If the *Handler Message Queue* contains a message for the current request, it is processed by the *SDD Proxy*, the request in question is redirected to the new data source and the message gets removed from the *Handler Queue*. To avoid bottlenecks, there can be multiple proxies where each of them is being managed via the *SDD Proxy APIs* by the *SDD Manager*.

SDD Manager

The *SDD Manager* acts as the central management component of the framework and provides the *SDD API* for overall framework control. To activate the framework a user invokes the *SDD API* with the following parameters: (i) a set of *Triggers* as well as a (ii) *Confidence Interval* to configure the Confidence Elasticity of the Framework. The *SDD Manager* then starts the first *SDD Proxy* and starts monitoring the average request rates as well as the utilization of the proxy via the *SDD Proxy API*. If the *SDD Manager* detects a potential bottleneck, it starts another proxy (additional ones if necessary based on the average request rate). The next task of the *SDD Manager* is to submit the provided *Triggers* to the *Analyzer Manager*, which uses them to invoke the corresponding request monitoring. A *Trigger* is used in the analyzer to decide whether a request needs to be handled or not. *Triggers* can, for example, be time based, size based or follow a customizable cost function and provide a threshold for triggering a handling action. The *Analyzer Manager* uses different pluggable *Analyzer Strategies* in correspondence with these submitted *Triggers* to determine if a request needs to be processed. If this is the case, the *Analyzer Manager* invokes the *Migration Manager* via the *Migration API* and provides the corresponding request including the results of its analysis. The *Migration Manager* is responsible for determining the potential *Migration Strategies* for the data resource in question. To achieve this it first contacts the *Dependency Manager*, which uses a dependency resolution mechanism to determine the corresponding *DAU* for the *DAI* being requested by the current request. The *Dependency Manager* in turn is tightly integrated with the *Security Manager*, which ensures all security constraints that are defined in the *DAUs* are satisfied before returning the results of the resolution. Once the dependency resolution has provided the *DAU*, it is analyzed by the *Migration Manager*. Specifically, it checks the results provided by the *Analyzer Manager* like request time, data size or a respective cost function against

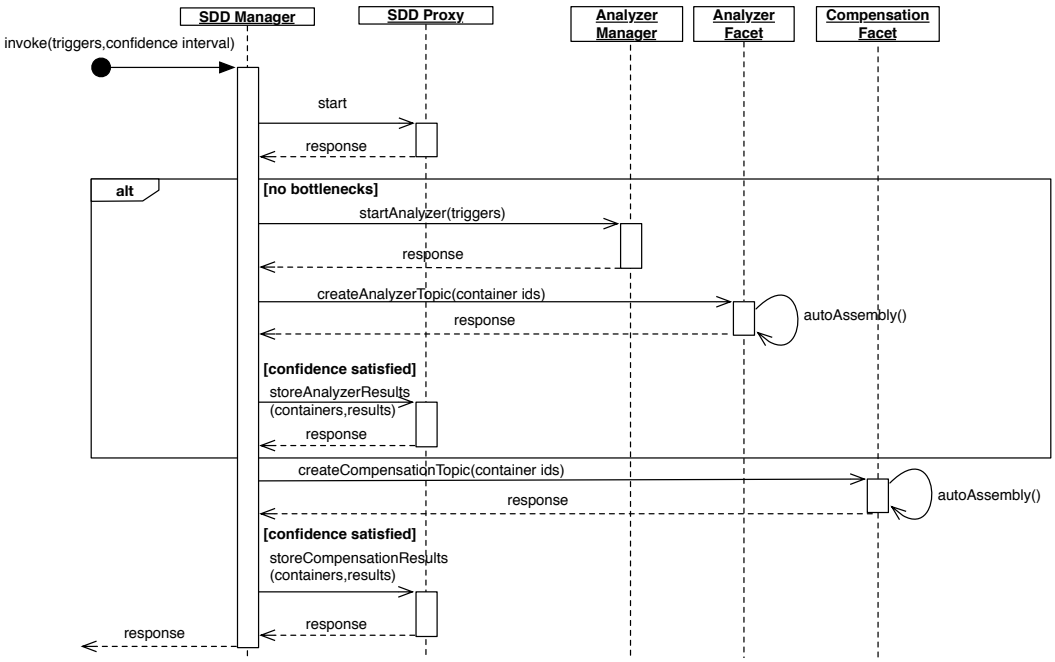

Figure 4 **SDD Manager sequence diagram.**

the attributes of the specific `view` being requested. It analyzes the update frequency as well as update sizes to determine if a migration should be performed as well as to to determine the fitting *Migration Strategy* and to execute it if applicable. Once the migration is finished the *Migration Manager* executes two tasks. First, it adds the request to the migrated data source to the *Handler Queue* including the new target (*DAI*) after the migration. This in turn triggers the corresponding *SDD proxies* to execute a redirection. Second, it registers the resource at the *Update Manager*, which in turn determines the fitting *Update Strategy* for the migrated data source to ensure that the data stays up to date. Once these steps are successfully finished, the *Migration Manager* updates the *DAIs* and *DAUs* to reflect the changes caused by the migration via the *Repository Manager*. A corresponding sequence diagram illustrating this process is shown in Fig. 4.

Analyzer Manager

The role of the *Analyzer Manager* is to determine if a request is a potential candidate for a migration. It watches the *Analyzer Queue* for requests that correspond to any of the previously provided *Triggers*. A *Trigger* is a threshold that matches an attribute that is the result of an *Analyzer Strategy*. *Analyzer Strategies* in turn are pluggable mechanisms that analyze a specific request based on the type of the request in question. Basically, we distinguish three different types of strategies: *Time Analyzers*, which determine the response time for a specific request; *Size Analyzers*, which determine the size of a request and response; as well as *Frequency Analyzers*, which determine the frequency of a request to a certain source. Additionally, there is also the ability to provide *Cost Function Analyzers*, which allow to integrate arbitrary cost functions and enable a much greater analytical

flexibility. Once a threshold is met, the *Analyzer Manager* submits the request in question including the results of the specific strategy to the *Migration Manager*.

Migration Manager

The *Migration Manager* is responsible for deciding if a data resource should be migrated based on the results of the *Analyzer Manager*. It is invoked via the *Migration API* with a specific request augmented with the results from the corresponding *Analyzer Strategy*. The *Migration Manager* then forwards this request to the *Dependency Manager*, which first determines the *DAI* that belongs to the requested data resource. Based on this *DAI* the *DAU* is determined only if all security constraints are being met, which in turn is ensured by the *Security Manager*. The retrieved *DAU* provides the foundation for deciding if a data source can and should be migrated. To achieve this, the *Migration Manager* relies on pluggable strategies that determine if a migration is feasible and possible. Such a *Migration Strategy* receives the retrieved *DAU* as well as the results from the *Analyzer Manager*. Based on this it does two things: first, it determines if a migration is possible by checking the results of the *Analyzer Manager* (e.g., response time, average transfer size) against the *updateFrequency* elements of the *DAU* as well as the constraints of the current infrastructure. If the result of this analysis leads to the conclusion that a migration is possible and feasible the *Migration Strategy* returns according results augmented with a *confidence value*. Second, the *Migration Strategy* provides a method to execute the actual migration. The framework is flexible regarding the specific migration mechanism and regards this as the responsibility of the strategy itself. One possible variant is the utilization of the *Smart Fabric Framework* (*Schleicher et al., 2015a*) since its provides an optimal foundation for infrastructure agnostic deployments (hence migrations) and supports the extended system model. In case of a *Smart Fabric Strategy* the *Infrastructure Specifications (IS)* are taken into account when deciding if a migration is feasible. This means the *Migration Strategy* can check which non functional characteristics apply and can incorporate them in the decision to migrate. Additionally, the execution of said migration is started by issuing a transfer requests to the *Smart Fabric Framework*. Based on the previously introduced confidence elasticity mechanism the *Migration Manager* then executes the corresponding *Migration Strategy* or in case none is found relies on a human interaction to perform the migration. Once the migration is finished, the *Migration Manager* creates new corresponding *DAIs* to reflect the migrations and updates the corresponding *DAUs*. Futhermore, it publishes a message to the *Handler Queue* and by doings so prompts the *SDD Proxy* to execute a redirection to the migrated data source. Once this is done the *Migration Manager* ensures the migrated data source is updated by registering the *DAIs* at the *Update Manager*.

Update Manager

The role of the *Update Manager* is to ensure that migrated data sources stay up to date if the original source is changed. To enable this it relies on pluggable *Update Strategies* and we basically distinguish the following different types: *Simple Copy Strategies*, which copy either a fraction or the entire data source; *Script Update Strategies*, which apply a more complex update strategy based on a script (e.g., rsync, sql scripts); as well as *Streaming Replication*

Strategies for continuous updates. These strategies again utilize the *Confidence Elasticity* mechanism by providing a confidence value. Based on the initially provided confidence interval of the framework the *escalation mechanism* selects an applicable *Update Strategy* or in the case none is found relies on a human interaction to perform the update. Once the update is finished the *Update Manager* updates the corresponding *DAI* via the *Repository Manager*.

Dependency Manager

The *Dependency Manager* is responsible for resolving unit dependencies between the modeled data entities, as described in the system model above. To achieve this, the dependencies between data entities in the system model are represented as a tree structure. Based on this tree structure the *Dependency Manager* creates a root node for each *DAU*. It then creates a corresponding leaf node for each *DAI* that is referenced in the dependency section of the *DAU*. After this it checks the related *DIs* and adds them as leaves. For every step the *Dependency Manager* also ensures that all security constraints are being met by checking the specific *DAU* with the *Security Manager*. If access is not permitted, the resolution is not successful.

Security Manager

The *Security Manager* is responsible for ensuring that all security constraints that apply to a given *DAU* are being met. To do so it checks to facets of a *DAU*. First, it ensures that a *DAU* description can be accessed by checking the `securityConstraints` element in the `metainformation` section. Second, it ensures that each view can be accessed as well as migrated by checking the corresponding `securityConstraints` element in the `view` sections of the *DAU*. To enable an open and evolvable security system, the *Security Manager* relies on pluggable *Security Strategies*. Examples of such strategies are *LDAP* or *OAuth* or other approaches like *RBAC* (*Hummer et al., 2013*), but can be extended to any other suitable security mechanism.

Repository Manager

The *Repository Manager* provides repositories for *DAUs* and *DAIs* and acts as a distributed registry keeping track of specific deployments and participating entities. It is responsible for storing and retrieving the system model. It manages two distinct system model repositories utilizing distributed key value stores, which store the JSON-LD files that represent *DAUs* and *DAIs* in a structured way. The *Repository Manager* provides a service interface to access these files as well as a search interface to query *DAUs* and *DAIs* based on specific elements. Additionally, it is responsible for managing dependencies between *DAUs* as well as *DAIs* and it seamlessly integrates with *Repository Managers* of other SDD Framework deployments ensuring a complete *DAU* and *DAI* lookup.

Implementation

For evaluation purposes we created a proof of concept prototype of our framework based on a set of RESTful microservices implemented in Ruby and packaged as Docker (https://www.docker.com/) containers. Every component that exposes a service interface

relies relies on the Sinatra (http://www.sinatrarb.com/) web framework. To enable the message-based communication for the *Analyzer, Handler* and *Update queues* we used RabbitMQ (https://www.rabbitmq.com/) as message-oriented middleware.

The *Repository Manager* utilize MongoDB (https://www.mongodb.org/) as its storage backend, which enables a distributed, open, and extendable key value store for the *DAU* and *DAI* repositories and provides the foundation for the distributed registry. The *SDD Proxy* was implemented as WEBrick (https://ruby-doc.org/stdlib-2.0.0/libdoc/webrick/rdoc/WEBrick.html) proxy server. Additionally, we patched the default Ruby http class in our prototype implementation to enable the transparent proxy behavior.

We implemented the *Analyzer Manager* with two *Analyzer Strategies*. Specifically, we implemented a *Web Request Response Time Analyzer* as well as *Web Request Response Size Analyzer*, which allowed us to analyze response times as well as average sizes of requests and responses. The *Migration Manager* was implemented with two *Migration Strategies*. The first strategy was a *MongoDB Strategy* supports the migration of MongoDB databases and collections, the second one was a *Docker Strategy* that enables a docker based container migration. For the *Update Manager* we reused the *MongoDB Strategy* as *MondoDB Database Copy Strategy* and *MondoDB Collection Copy Strategy* and additionally implemented a file based *SCP Full Copy Strategy*, which transfers files via Secure Copy (`scp`).

The prototype implementation is available online and can be found at https://bitbucket.org/jomis/smartdata.

EVALUATION

Setup

As basis for our evaluation we used the URBEM Smart City Application (USCA) (*Schleicher et al., 2016c*), a holistic interdisciplinary decision support system, which has been used for city infrastructure planning tasks especially in the context of energy and mobility systems. We choose the USCA, because it represents an optimal candidate for our evaluation due to the following characteristics: (i) It heavily relies on a diverse set of data sources, where most of them belong to stakeholders and are under strict security and compliance regulations; (ii) It is an application that has to deal with changing requirements that make it necessary to incorporate new data sources dynamically; (iii) Due to the nature of the application as a planning tool for energy and mobility systems it is a common case to incorporate data sources from other cities around the globe.

Due to the strict data security regulations, we were not allowed to use the original data from the URBEM domain for our evaluation scenario. To overcome this limitation, we created anonymized random samples of the most common data sources that are being used in the USCA. The specific datasets we used included building data with different granularity levels, thermal and electrical network data as well as mobility data. Based on these data sources we created *Data Units (DAUs)* for each of them as well as exemplary data services as *Technical Units (TUs)*. As a next step we looked at the common request patterns for these types of data based on the request history of the USCA. Since the USCA

relies on user input via its graphical user interface, we created sample clients, which tested these request patterns in order to enable automated testing. Based on these foundations we created two different evaluation scenarios.

For the first scenario, we provisioned two VM instances in our private OpenStack cloud, each with 7.5GB of RAM and 4 virtual CPUs. These two instances represented Data Centers in the cities of Vienna and Melbourne. In order to simulate realistic transfer times between these two different regions we used Linux Advanced Traffic and Routing (`tc`) to simulate the average delay between these regions including common instabilities. Each of these instances was running Ubuntu 16.04 LTS with Docker.

For the second scenario we provisioned three VM instances in the Google Cloud Platform. We used `n1-standard-2` instance types each with 7.5GB of RAM and 2 virtual CPUs. In order to get a realistic geographic distribution, we started each of these instances in different cloud regions. Specifically, we started one in the `us-central` representing the city of Berkeley, one in the `europe-west` region representing the city of Vienna and one in the `asia-east` region representing the city of Hong Kong. Each of these instances was again running Ubuntu 16.04 LTS with Docker.

For monitoring purposes we used Datadog (https://www.datadoghq.com/) as monitoring platform. We submitted custom metrics for request and response times, and monitored system metrics for bytes sent as well as overall instance utilization to ensure over utilization had no impacts on our results.

Experiments

In this section we give a detailed overview of the conducted experiments within the two scenarios.

Scenario One

In the first scenario we wanted to evaluate the impact of SDD in the context of a simple scenario using USCA for analytics of two cities. We simulated a case in which stakeholders use USCA to compare the impact of city planning scenarios on the building stock, thermal network and public transport between the city of Vienna and Melbourne.

To achieve this we generated 10 data services based on the *DAU* we defined for buildings, networks and mobility as well as the corresponding *DAIs* and deployed them as docker containers on the instance representing Melbourne. We did the same for the instance in Vienna. In the next step we deployed 5 clients simulating the previously mentioned request patterns on the Vienna instance. This setup of clients and sources represented a common sample size of clients and services used in the current USCA context. As a last step, we deployed the SDD framework, specifically three containers: one for the *SDD proxy*, one for the *Repository Manager* and one for the other components of the *SDD Framework*. An overview of this evaluation scenario can be seen in Fig. 5.

In the context of this scenario we distinguished 4 different request types to three different kinds of *DAUs*. The first *DAU*, *buildings* represented a larger data source (158.000 entities) with low update frequency (once a year). For this resource, we had two request types on two views of this resource; specifically, */buildings* and */blocks* representing two different

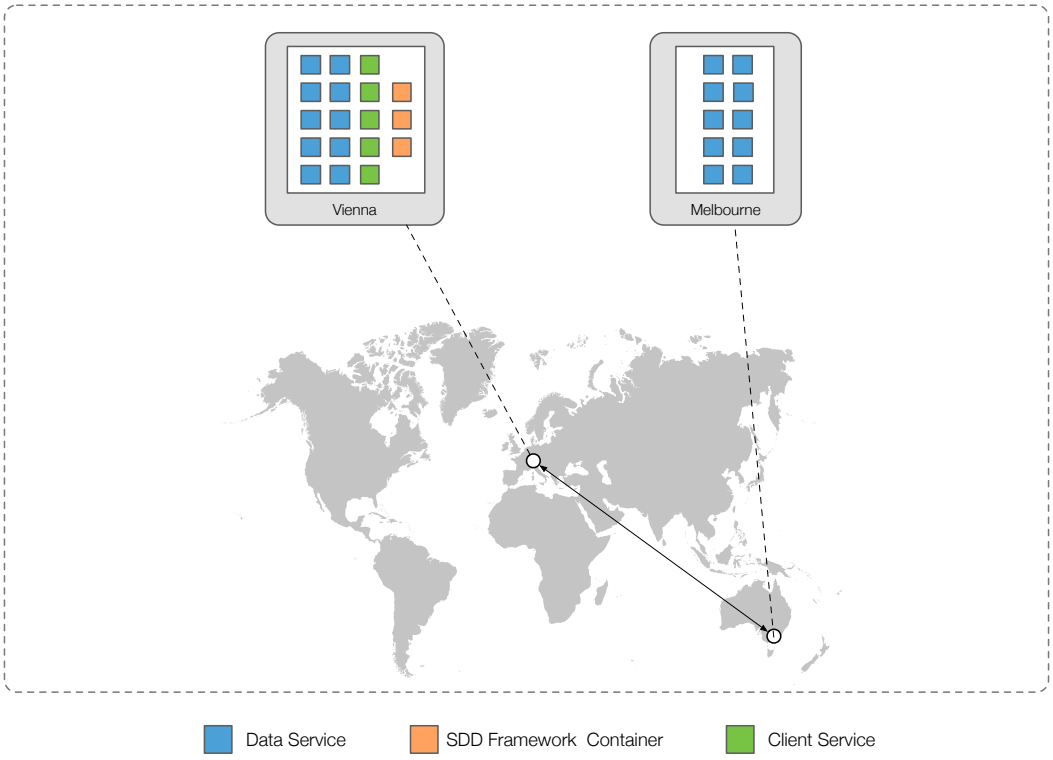

Figure 5 **Evaluation setup for Scenario One.**

levels of detail. The second *DAU* was *networks* again representing a larger data source (68.000 entities) with low update frequency (once quarterly). For this resource we had one request type on the only view of this resource, namely */networks*. The last *DAU* was *mobility* representing a smaller data source (4.000 entities) with high update frequency in the public transport context.

To establish a baseline, we started our evaluation with the *SDD Framework* deactivated and monitored the response times of each of these four request types, as well as the transferred bytes from the Melbourne instance through custom Datadog monitors. After 5 min we started the *SDD Framework* by submitting a request to the *SDD Manager* on the Vienna instance via the *SDD API* with *Triggers* for response times longer than 3 s and a confidence value that matched our automated *Migrations Strategies* since we wanted to test the automated capabilities of the *SDD Framework*. After submitting the request, we continued to monitor the results of the Datadog monitors over a course of additional 5 min.

The results of our evaluation can be seen in Fig. 6. In the figure, we see the different characteristics of the response times for the 4 request types. *Buildings*, *blocks* as well as *networks* with longer response times, *mobility* with a rather short response time. Given the submitted *Triggers* as well as the implemented *Migration Strategy* in context with size and update characteristics of the *DAUs*, *Buildings, blocks and networks* qualified for migration. We see that the framework correctly identified these requests and starts migrating the

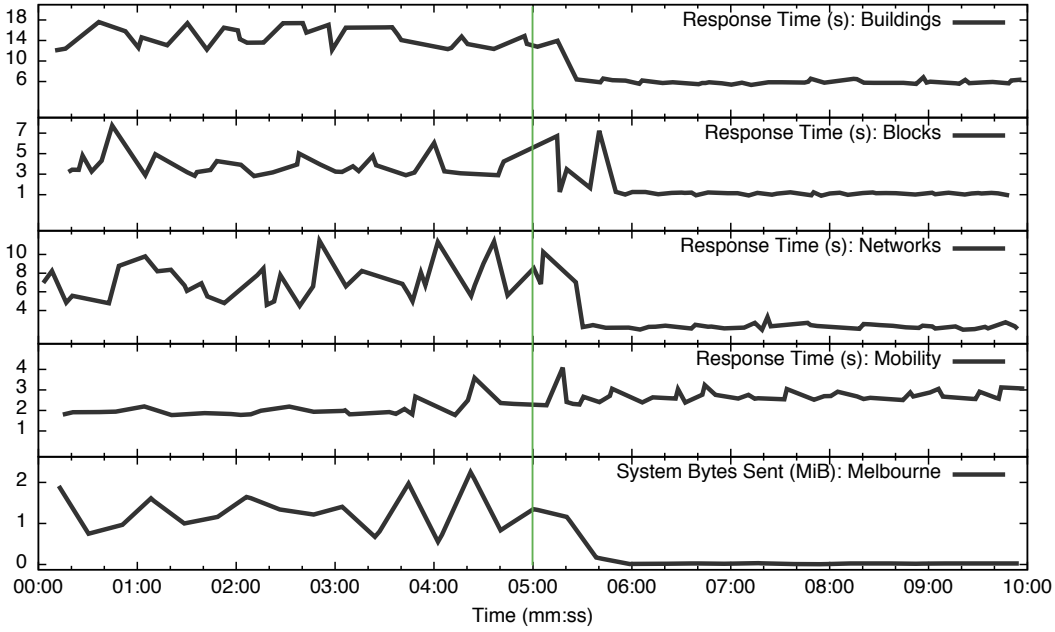

Figure 6 **Evaluation results for Scenario One.**

corresponding *DAIs*, around minute 5. The total migration time for all three resources was 59.3 seconds during this time the *SDD proxy* keeps forwarding the requests to the original *DAI*. After the migration has finished and the *SDD Proxy* successfully started redirecting to the new *DAIs*, we see a significant reduction in the average response times for all three request types. The *mobility* source was not migrated since it didn't qualify due to the fact that this resource had response times below the trigger. We also see that the response time for this resource shows an increase, which is due to the specific proxy implementation and caused by the overhead of redirection checks after the framework has been activated and is present for all request types. The efficiency of the specific proxy implementation was not focus of this work and does not influence the validity of the presented results, since the introduced overhead affects all requests equally. Additionally, we see that in this scenario the network transfer from the instance representing Melbourne was reduced by 97% after the migrations finished. This is due to the fact that only the *DAI* for mobility remains active on this instance and no other clients or framework components are actively sending data from Melbourne. The aggregated overview in Table 1 shows that average and median response times along with variance in response times for all migrated *DAIs* were reduced significantly.

Scenario Two

In the second scenario we wanted to evaluate the impact of SDD in the context of a larger and more complex scenario using USCA for analytics in an internet of cities setup with cities in different regions. We again simulated a case in which stakeholders use USCA to compare the impact of city planning scenarios on the building stock, thermal network and public transport. This time between the cities of Berkeley, Vienna and Hong Kong, which

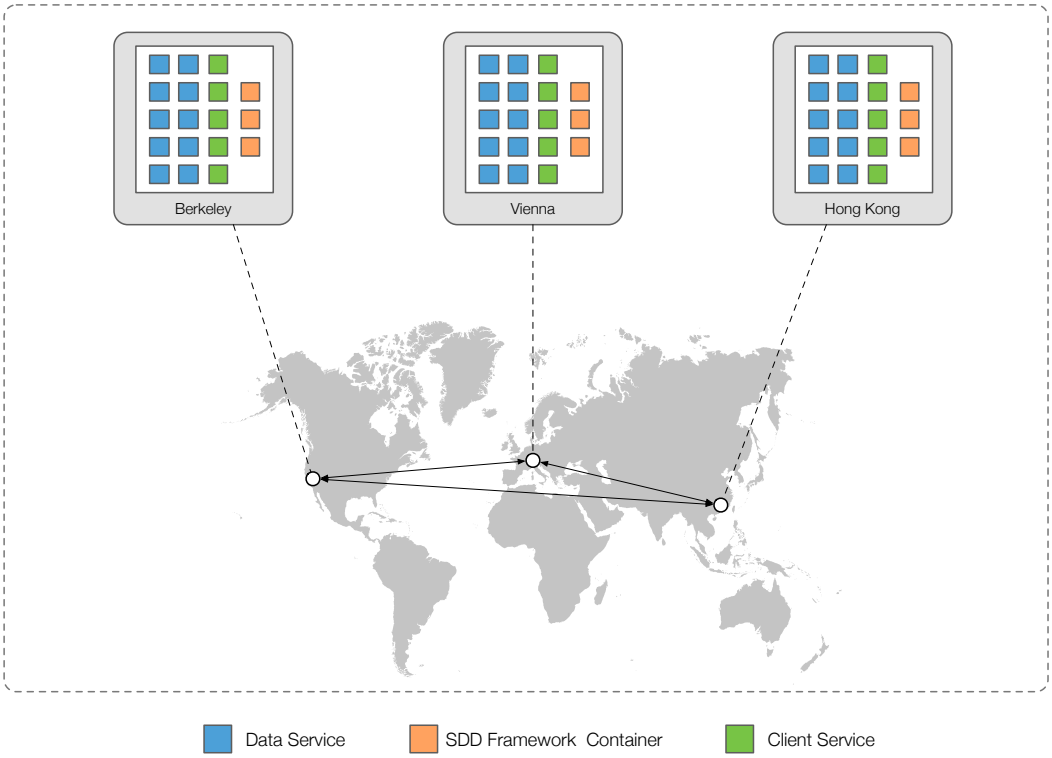

Figure 7 **Evaluation setup for Scenario Two.**

Table 1 **Average, median and standard deviation for response times per request type in Scenario One.**

Request type	Status	Average response time	Median response time	Standard deviation response time
Buildings	Inactive	14.561 s	14.272 s	1.857 s
	Active	5.94 s	5.848 s	0.354 s
Blocks	Inactive	3.833 s	3.425 s	1.316 s
	Active	1.122 s	1.154 s	0.102 s
Networks	Inactive	7.21 s	6.843 s	1.928 s
	Active	2.298 s	2.268 s	0.257 s
Mobility	Inactive	2.196 s	1.982 s	0.504 s
	Active	2.727 s	2.677 s	0.212 s

were placed in the respective regions of the Google Cloud platform as described in the setup section.

To achieve this, we again generated 10 data services based on the *DAU* we defined for buildings, networks and mobility as well as the corresponding *DAIs* and deployed them as Docker containers on all three instances. In the next step, we deployed 5 clients per city simulating the previously mentioned request patterns. We then deployed the SDD framework, specifically three containers: one for the *SDD proxy*, one for the *Repository Manager* and one for the other components of the *SDD Framework* on every instance. An overview of this evaluation scenario is depicted in Fig. 7.

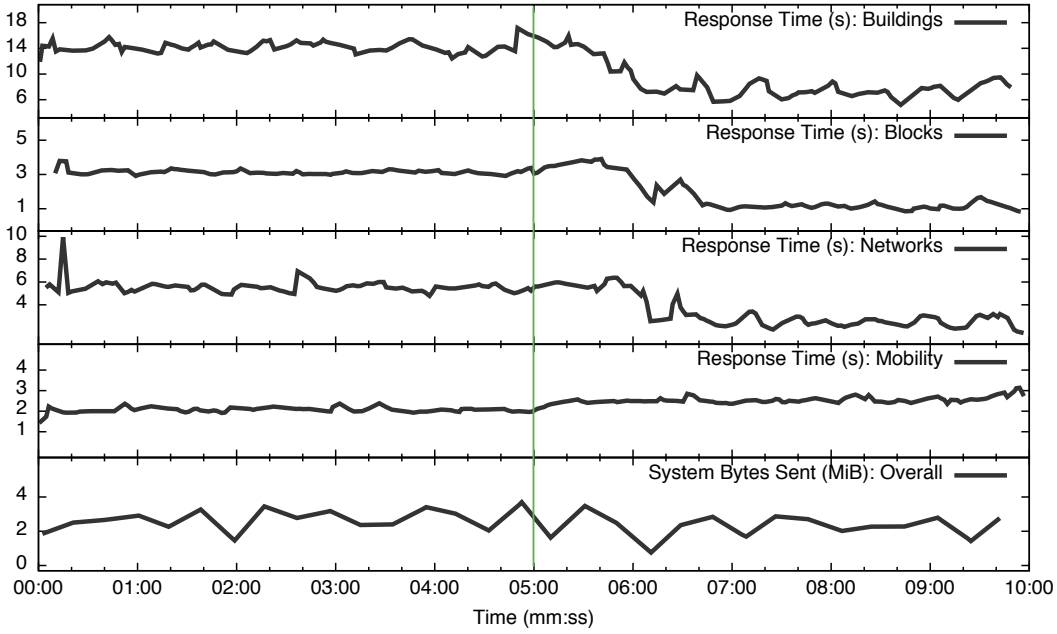

Figure 8 **Evaluation results for Scenario Two.**

In this scenario we distinguished the same 4 request types as before. To establish a baseline, we started our evaluation with the *SDD Framework* deactivated and monitored the response times of each of these four request types, as well as the transferred bytes from all participating instances through custom Datadog monitors. After 5 min we started the *SDD Framework* by submitting a request to the *SDD Manager* on each of the three instances via the *SDD API* with *Triggers* for response times longer than 3 s and a confidence value that matched our automated *Migrations Strategies* since our focus was again on testing the automated capabilities of the *SDD Framework*. After submitting the requests we continued to monitor the results of the Datadog monitors over a course of additional 5 min.

The results of our evaluation can be seen in Fig. 8. By investigating the Figure, we notice that the framework again correctly identified the three request types to *buildings*, *blocks* and *networks* as candidates for migrations. The framework starts migrating the corresponding *DAIs* around minute 5. In this more complex case the total migration time for all three resources over all three instances was 112.32 seconds. After the migration has finished and the *SDD Proxy* successfully started redirecting to the new *DAIs* we again see a significant reduction in the average response times for all three request types. In terms of network transfer we don't see a significant reduction as opposed to Scenario One since in this setup there are active clients, as well as framework components deployed on all instances that continue to issue requests, hence sending data. The aggregated overview in Table 2 shows that average and median response times for all migrated *DAIs* again were reduced significantly. In contrast to the lab environment of Scenario One, a significant reduction

Table 2 Average, median and standard deviation for response times per request type in Scenario Two.

Request type	Status	Average response time	Median response time	Standard deviation response time
Buildings	Inactive	14.214 s	14.266 s	0.98 s
	Active	7.825 s	7.613 s	1.335 s
Blocks	Inactive	3.203 s	3.142 s	0.246 s
	Active	1.273 s	1.161 s	0.37 s
Networks	Inactive	5.571 s	5.548 s	0.582 s
	Active	2.594 s	2.489 s	0.578 s
Mobility	Inactive	2.089 s	2.088 s	0.162 s
	Active	2.562 s	2.514 s	0.156 s

in response time variance was not observed, which can be attributed to the performance variability of cloud instances as well as the distribution over the chosen regions.

Our experiments showed that we could significantly reduce the response times and hence the QoS for the URBEM Smart City Application. Specifically, we showed a reduction of the response times by 66% on average over all three migrated request types. We also demonstrated that the framework was able to correctly identify the *DAIs* to be migrated utilizing the views specified in the corresponding *DAUs*. Finally, we showed that we could produce these results both in a laboratory setting as well as in a geographically dispersed cloud setup.

Threats to applicability

While the presented system model and framework fulfill the requirements set forth in the context of the previously introduced URBEM Smart City Application, certain threats to the general applicability of SDD remain. The initial evaluation setup for Scenario One used `tc` to introduce the delays between the two instances representing Vienna and Melbourne. It could be argued that this simulated setup was not representative for the evaluation. The fact that the experiments showed similar results in a globally distributed deployment refute this claim. Beyond this, the current evaluation relied on simulated clients and data sources for the experiments. To ensure that the used workloads and data sources are realistic and representative, we gathered workload patterns and anonymized example records from the URBEM smart city application used by domain experts.

RELATED WORK

The recent trend in smart city research towards the introduction of smart city platforms and reference models has been further fanned by the rise of the Internet of Things (IoT). While all of these approaches mention the importance of data management in the context of the massive amount of data, its heterogeneity and multitude of security and compliance constraints, it currently either is not a framework element or they do not provide specific solutions for this problem. *Chourabi et al. (2012)* present a framework to understand the concepts of a smart citie on a more abstract level. The authors introduce a conceptual framework to comprehend the vital elements in a smart city by identifying critical factors. In

the context of ICT, they identify security, privacy as well as accessibility as central elements for smart city applications. In a similar way, *Naphade et al. (2011)* present innovation challenges for smarter cities. The authors identify the management of information across all cities' information systems including the need for privacy and security as a central challenge in the success of smart cities underlining the importance of a data management approach like ours. On a more concrete level *Bonino, Pastrone & Spirito (2015)* present ALMANAC, a smart city platform with a focus on the integration of heterogenous services. They identify challenges smart city applications face, also in terms of infrastructure and specifically mention the importance of taking data ownership and exchange between the different smart city stakeholders into account. Compared to our approach, however, they do not provide a specific solution to address these challenges, especially none applicable to legacy data.

In the context of IoT where more and more data sources emerge, data management plays a central role. *Jin et al. (2014)* introduce a smart city IoT infrastructure blueprint. The authors focus on the urban information system starting from the sensory level up to issues of data management and cloud-based integrations. They identify key IoT building blocks for a smart city infrastructure, one of them being the so called Data-centric IoT, in which data management is a central factor. In a similar high level manner, *Petrolo, Loscrì & Mitton (2017)* introduce the VITAL platform as an IoT integration platform to overcome the fragmentation issue in smart cities. They mention data challenges that arise by introducing IoT and specifically underline the importance of privacy and security in this context. The need for data management in order to integrate the produced results also applies to a lot of other IoT platforms in order to make their results accessible for further analytics. Examples of such frameworks are *Chen & Chen (2012)*, who present a data acquisition and integration platform for IoT. A central element in their architecture is a contextual data platform that needs to integrate with multiple heterogenous data sources. *Cheng et al. (2015)* present CiDAP a city and data analytics platform based on SmartSantander (*Sanchez et al., 2014*) a large-scale testbed that helps with issues arising from connecting and managing IoT infrastructure. They name data management, especially the exchange of data as well as attached semantics as one central challenge. Finally, in the context of specific IoT smart city applications, *Kyriazis et al. (2013)* present two sustainable smart city applications in the transportation and energy domain. They clearly identify security in the context of data as a specific challenge in enabling these applications emphasizing the importance of approaches like ours. In the context of IoT platforms and IoT smart city applications our approach provides the missing link for both security aware data management within frameworks, tackling many of the identified challenges as well as providing an ideal way to expose the collected data in a usage-aware distributed way.

A vital element to enable this kind of data management in an efficient way is the ability to migrate data resources. In the context of said migration there are several approaches relevant in the context of our work. *Amoretti et al. (2010)* propose an approach that facilitates a code mobility mechanism in the cloud. Based on this mechanism, services can be replicated to provide a highly dynamic platform and increase the overall service availability. In a similar way, *Hao, Yen & Thuraisingham (2009)* discus a cost model and a genetic decision

algorithm that addresses the tradeoff on both service selection and migration in terms of costs, to find an optimal service migration solution. They introduce a framework for service migration based on this adaptive cost model. Opposed to our approach, they focus on the service migration aspect without explicitly addressing the data aspect and do not provide means for incorporating security and other characteristics on a more fine grained data level. In the context of potential migration strategies there are several interesting approaches. *Agarwal et al. (2010)* presents Volley, an automated placement approach for distributed cloud services. Their approach uses logs to derive access patterns and client locations as input for optimization and hence migration. *Ksentini, Taleb & Chen (2014)* introduce a service migration approach for follow me clouds based on a Markov Decision Process (MDP). In a similar way, *Wang et al. (2015)* demonstrate a MDP as a framework to design optimal service migration policies. All these approaches can be integrated as potential migration strategies in our approach.

CONCLUSION

Current smart city application models assume that produced data is managed by and bound to its original application. Such data silos have emerged, in part due to the complex security and compliance constraints governing the potentially sensitive information produced by current smart city applications. While it is essential to enforce security and privacy constraints, we have observed that smart city data sources can often provide aggregated or anonymized data that can be released for use by other stakeholders or third parties. This is especially promising, as such data sources are not only relevant for other stakeholders in the same city, but also other smart cities around the globe. We argue that future smart city applications will use integrated data from multiple sources, gathered from different cities to significantly improve efficiency and effectiveness of city operations, as well as citizen wellbeing. To allow for the creation of such applications, a seamless and efficient mechanism for description and access of available smart city data is required.

In this paper, we presented Smart Distributed Datasets (SDD), a methodology and framework to enable transparent and efficient distributed data access for data sources in smart city environments. A system model that provides a simple abstraction for the technology-agnostic description of available data sources and their subsets was introduced. Subsets can represent different aspects and granularities of the original data source, along with relevant characteristics common in the smart city domain. Based on this abstraction, we presented the SDD framework, a middleware toolset that enables efficient and seamless data access for smart city applications by autonomously relocating relevant subsets of available data sources to improve Quality of Service (QoS) based on a configurable mechanism that considers request latency, as well as costs for data transfer, storage, and updates. We evaluate the presented framework using a case study in the context of a distributed city infrastructure decision support system and show that selective relocation of data subsets using the SDD framework can improve QoS through significantly reducing response times by 66% on average.

In our ongoing work, we will integrate additional optimization mechanisms in the data migration process to further improve framework performance. We also plan to more

closely integrate the SDD framework with our overall efforts in designing, engineering, and operating smart city applications (*Schleicher et al., 2015b*). Furthermore we will create additional smart city applications, covering different application areas in collaboration with domains experts from URBEM, as well as other smart city initiatives. In the context of our research on the future Internet of Cities, we will extend the SDD framework to support autonomous, ad-hoc coordination of globally distributed SDD proxies to further optimize *DAI* placement in smart city application ecosystems.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The research leading to these results has received funding from the URBEM doctoral college. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
URBEM doctoral college.

Competing Interests

Schahram Dustdar is an Academic Editor for PeerJ.

Author Contributions

- Johannes M. Schleicher conceived and designed the experiments, performed the experiments, analyzed the data, wrote the paper, prepared figures and/or tables, performed the computation work, reviewed drafts of the paper.
- Michael Vögler and Christian Inzinger conceived and designed the experiments, analyzed the data, wrote the paper, prepared figures and/or tables, reviewed drafts of the paper.
- Schahram Dustdar reviewed drafts of the paper.

Data Availability

The following information was supplied regarding data availability:

Source Code Repository for the Prototype Implementation: https://bitbucket.org/jomis/smartdata.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.115#supplemental-information.

REFERENCES

Agarwal S, Dunagan J, Jain N, Saroiu S, Wolman A, Bhogan H. 2010. Volley: automated data placement for geo-distributed cloud services. In: *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2–2.

Amoretti M, Laghi MC, Tassoni F, Zanichelli F. 2010. Service migration within the cloud: code mobility in SP2A. In: *Proceedings of the international conference on high performance computing & simulation*. Piscataway: IEEE, 196–202 DOI 10.1109/HPCS.2010.5547130.

Bonino D, Pastrone C, Spirito M. 2015. Towards a federation of smart city services. In: *Proceedings of the international conference on recent advances in computer systems*, 163–168.

Chen YS, Chen YR. 2012. Context-oriented data acquisition and integration platform for Internet of Things. In: *Proceedings of the conference on technologies and applications of artificial intelligence*. Piscataway: IEEE, 103–108.

Cheng B, Longo S, Cirillo F, Bauer M, Kovacs E. 2015. Building a big data platform for smart cities: experience and lessons from Santander. In: *Proceedings of the international congress on big data*. Piscataway: IEEE, 592–599.

Chourabi H, Nam T, Walker S, Gil-Garcia JR, Mellouli S, Nahon K, Pardo TA, Scholl HJ. 2012. Understanding smart cities: an integrative framework. In: *Proceedings of the 45th Hawaii international conference on system sciences*. Piscataway: IEEE, 2289–2297.

Hao W, Yen I-L, Thuraisingham B. 2009. Dynamic service and data migration in the clouds. In: *Proceedings of the 33rd IEEE international computer software and applications conference*. Piscataway: IEEE, 134–139.

Hummer W, Gaubatz P, Strembeck M, Zdun U, Dustdar S. 2013. Enforcement of entailment constraints in distributed service-based business processes. *Information and Software Technology* **55(11)**:1884–1903 DOI 10.1016/j.infsof.2013.05.001.

Inzinger C, Nastic S, Sehic S, Vögler M, Li F, Dustdar S. 2014. MADCAT—a methodology for architecture and deployment of cloud application topologies. In: *Proceedings of the international symposium on service-oriented system engineering*. Piscataway: IEEE, 13–22.

Jin J, Gubbi J, Marusic S, Palaniswami M. 2014. An information framework for creating a smart city through internet of things. *IEEE Internet of Things Journal* **1(2)**:112–121 DOI 10.1109/JIOT.2013.2296516.

Ksentini A, Taleb T, Chen M. 2014. A Markov Decision Process-based service migration procedure for follow me cloud. In: *Proceedings of the international conference on communications*. Piscataway: IEEE, 1350–1354.

Kyriazis D, Varvarigou T, White D, Rossi A, Cooper J. 2013. Sustainable smart city IoT applications: heat and electricity management amp; eco-conscious cruise control for public transportation. In: *Proceedings of the 14th international symposium on "A World of Wireless, Mobile and Multimedia Networks"*. Piscataway: IEEE, 1–5 DOI 10.1109/WoWMoM.2013.6583500.

Naphade M, Banavar G, Harrison C, Paraszczak J, Morris R. 2011. Smarter cities and their innovation challenges. *Computer* **44(6)**:32–39 DOI 10.1109/MC.2011.187.

Newman S. 2015. *Building microservices*. Sebastopool: O'Reilly Media, Inc.

Petrolo R, Loscrì V, Mitton N. 2017. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies* **28**:e2931 DOI 10.1002/ett.2931.

Sanchez L, Muñoz L, Galache JA, Sotres P, Santana JR, Gutierrez V, Ramdhany R, Gluhak A, Krco S, Theodoridis E, Pfisterer D. 2014. SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks* **61**(**November**):217–238 DOI 10.1016/j.bjp.2013.12.020.

Schleicher JM, Vögler M, Dustdar S, Inzinger C. 2016a. Enabling a smart city application ecosystem: requirements and architectural aspects. *IEEE Internet Computing* **20**(**2**):58–65.

Schleicher JM, Vögler M, Inzinger C, Dustdar S. 2015a. Smart fabric–an infrastructure-agnostic artifact topology deployment framework. In: *Proceedings of the international conference on mobile services*. Piscataway: IEEE, 320–327.

Schleicher JM, Vögler M, Inzinger C, Dustdar S. 2015b. Towards the internet of cities: a research roadmap for next-generation smart cities. In: *Proceedings of the international workshop on understanding the city with urban informatics*. New York: ACM, 3–6.

Schleicher JM, Vögler M, Inzinger C, Dustdar S. 2016b. Smart brix—a continuous evolution framework for container application deployments. *PeerJ Computer Science* **2**:e66 DOI 10.7717/peerj-cs.66.

Schleicher JM, Vögler M, Inzinger C, Fritz S, Ziegler M, Kaufmann T, Bothe D, Forster J, Dustdar S. 2016c. A holistic, interdisciplinary decision support system for sustainable smart city design. In: *Proceedings of the international conference on smart cities*. Cham: Springer, 1–10.

Vögler M, Schleicher JM, Inzinger C, Dustdar S, Ranjan R. 2016. Migrating smart city applications to the cloud. *IEEE Cloud Computing* **3**(**2**):72–79.

Wang S, Urgaonkar R, Zafer M, He T, Chan K, Leung KK. 2015. Dynamic service migration in mobile edge-clouds. In: *Proceedings of the 14th IFIP networking conference*, 1–9.