A Distributed Generic Data Structure for Urban Level Building Data Monitoring

Stefan Glawischnig, Harald Hofstätter, Ardeshir Mahdavi

A Distributed Generic Data Structure for Urban Level Building Data Monitoring

Stefan Glawischnig, Harald Hofstätter, and Ardeshir Mahdavi

Vienna University of Technology
Department for Building Physics and Building Ecology
Karlsplatz 13, 1040 Vienna, Austria
{stefan.glawischnig,harald.e259.hofstaetter,
ardeshir.mahdavi}@tuwien.ac.at

Abstract. Building a generic data structure that handles building realated data at an urban scale offers certain challenges. Real world entities must be captured in an environment that allows for the communication of relevent data. The associated software components must be maintainable and reliable. The present contribution describes efforts to enhance a well tested building monitoring framework to handle building data at an urban scale. This requires the development of a distributed, generic and enhancable data store, as well as the conceptualization of a modular and scalable application architecture. The scalable data store is introduced, as well as the modularization process of the application logic, including data handling and communication routines. Furthermore, the concept of Virtual Datapoints and Virtual Datapoint Collections enables urban entities (for instance buildings) to communicate their status to the system in an effective way.

Keywords: Urban Monitoring, Generic Data Structure, Distributed Systems

1 Introduction

Urbanization and sustainability strategies (e.g. Europe 2020) raise the need to address ecologic and economic issues on an urban level. A building is not to be recognized as a discrete entity but as part of a dynamic system, that interacts with it's surroundings [1]. The urban environment produces massive amounts of data, not all of them related to buildings, but nevertheless influential. For instance, it is common practice to integrate weather forecasts and historical weather data into a building's Energy Management and Control System (EMCS) as well as to utilize it as base data for light and thermal simulation [2][3]. Pang et al. introduced a framework to compare the building performance with predictive values of an EnergyPlus simulation in real time [4]. The EMCS transmits data to the simulation via a BACnet interface. The presented approach offers a promising solution for accessing EnergyPlus models in real time, but the resulting data collection is very specific and task oriented.

Previous research by the authors concentrated on the development of a vendor and platform independent building monitoring system [5][6]. The motivation was to develop a storage concept that handles building related data in an unified way, regardless of the initial data format or producer. Nevertheless, a majority of research projects focused on various stages of a building's life cycle and considered buildings as singular entities. For instance, the SEMERGY project developed a concept to help remove data-related disincentives and to facilitate the integration of building performance evaluation in building design and retrofit [7]. On a lower application level, monitoring related projects elaborated a generic data structure that is used in a number of simulation applications and building data representations.

1.1 Monitoring System Toolkit

The Monitoring System Toolkit (MOST) was designed to offer platform and vendor independent access to building data and follows a distributed and service oriented approach [9]. It offers the possibility to integrate third party data (EMCS, BMS, etc.) by implementing a connector interface [8]. The toolkit consists of various decoupled services (Fig. 1).

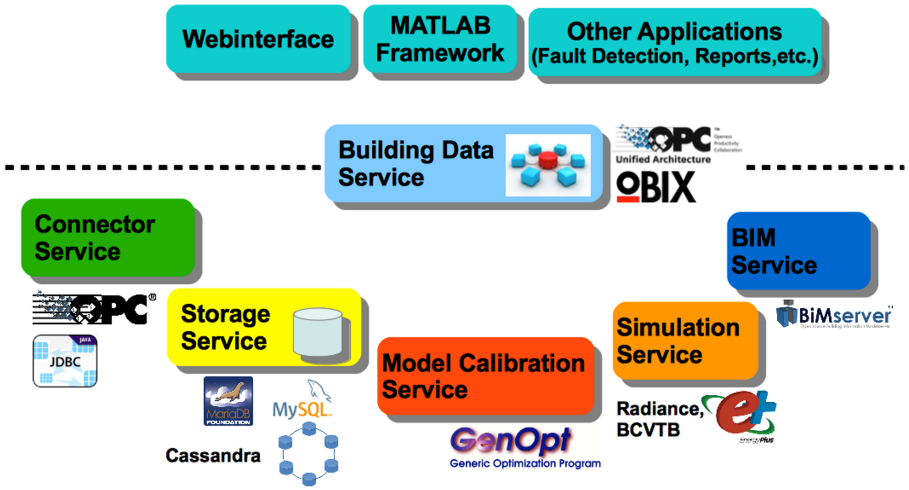

Fig. 1. Monitoring system service structure

A building data service establishes connections between client applications and the system backend. To handle input data sources generically, all data is organized by datapoints and zones. A datapoint is mapped to any data producing entity, for instance a physical sensor, an entire building's accumulated

energy demand or a person. Zones define administrative entities that establish location contexts for datapoints. Data might be added to the data store via three methods: a connector implementation, the service layer or a virtual datapoint [10]. The connector service introduces a vendor independent communication process with common building management systems and building automation technologies. Furthermore, connector implementations enable the integration of historical datasets, for instance weather data that was generated by a CSV export. Beside connectors, the framework's functionality is exposed via two standard industry protocols, OPC Unified Architecture, and Open Building Information Xchange (oBIX). Additionally, a custom REST and RPC implementation offer data access. Entities that access the service layer always read or write one specific datapoint value at a time. The initial data store consists of a relational database (MySQL) that defines entities for datapoints and zones with the respective configurations, as well as an data entity that stores the imported or recorded values. A data tuple has the following form: $< tuple\_id, timestamp, value, datapoint\_ref >$.
Data access (read, write) as well as data processing routines (calculating periodic values) are encapsulated in database procedures. To minimize the security risk of SQL injections, database access is restricted to these procedures. To fit different use cases data replication was introduced. The partitions on the master machine were optimized to provide live data access. Two slave machines handeled batch processing jobs and long term, historic data access. The database performance was evaluated by running three testcases that consisted of two scenarios (Tab. 1)

Table 1. Database performance test scenarios

Test scenario	#Datapoints	#Zones	#Values
A	$10*10^3$	$2.5*10^3$	$2.5*10^6$
B	$10*10^4$	$2.5*10^4$	$>2.5*10^6$

Test case #1 evaluated the write performance of the system. Five concurrent connections are established and scenario A and B executed. Test case #2 analyzes the data retrieval performance of scenario A and B by calling various data processing routines. The third test case consists of an iteration of the same read operations as test case #2, as new values are simultaneously added every 50ms. The benchmarking results showed no significant performance bottlenecks. For instance, test case #1 performed a maximum write throughput of 1000 values per second. A thorough documentation of the performance tests can be found at [11].

To test the suitability of the application and data store architecture at an urban scale five buildings were concurrently administrated by one framework instance. Physical sensor's recording intervals are specified in seconds and can theoretically be defined as $r = [1, 1844674407e + 19]$ for r $\in \mathbb{N}$ sec. Practically

sensor recording intervals $r = [3600, 86000]$ sec. This corresponds to a range between one hour and an entire day. Currently, the database handles $8.0*10^4$ transactional commits, $1.3*10^5$ inserts, $7.0*10^5$ selects and $5.3*10^5$ updates per day. Managing multiple buildings simultaneously revealed performance problems that were not identified by the performance tests. Adding and retrieving data, especially functions that conduct calculations, resulted in severe performance incursions and eventually even crashed the system. Aside the given technical limitations the lack of framework modularization appeared as an conceptual limitation.

1.2 Approach

The initial framework implementations were intended for single building use and did not support several buildings or even entire cities. Although it followed a web-based approach with well defined layers, the framework was only deployable as one compound application, entailing problems both in scalability and reliability. To provide an environment that operates at an urban scale and thus allows simultaneous handling of buildings, as well as the establishment of relationships between buildings and building data required the solution the following conceptual problems:

- Optimize and generalize the data store
- Modularize the application into standalone components
- Establish a data communication process between components

2 Data Storage

The database was designed to offer a generic data store for one building. Sensor measurements as well as other building related data (for instance user data, sensor configurations, datapoint definitions, etc.) are stored in one central database. Experience showed that up to 90% of the database-induced network traffic was caused by read and write operations on the measurement data table. Resolving this issue was established by these steps:

- Seperating the highly dynamic data entities from the configuration data
- Move data processing routines to a higher application layer to allow parallelization at software level

As can be seen in Fig. 2 four objects show a highly dynamic behaviour: datapoint, data, zone and warning. A datapoint maps a physical entity, be it a sensor, a person or a calculated aggregation object (e.g. accumulated temperature of a room). A zone administrates datapoints and handles access right management. A zone might refer to a room or a floor. The data table holds all imported and recorded data tuples. If an error occurs (for instance a broken batch process, or a broken sensor) it is written into the warning table.

To store these data generically, multiple relational and non-relational data stores were analyzed regarding their performance, scalability and data model. Following requirements were elaborated to ensure scalability:

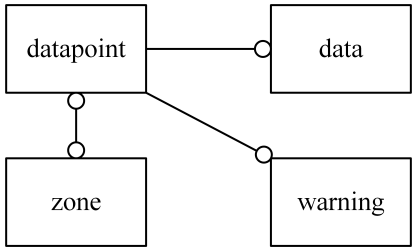

Fig. 2. High-traffic producing database entities. Excerpt from the ER diagram.

- Ensure transaction security for sensitive master data (for instance configurations, user information)
- Flexible and extensible data model
- High read and write performance for the datapoint value data
- High read performance for the master data store

Due to potential security risks [12] of non-relational stores and the fact that relational databases are well established in productive environments the master data should be kept in a relational environment. Transaction security is not a crucial condition for storing datapoint values. In practice, sensors send a vlaue every minute or even second. In case a measurement gets lost, the effect on an entire day's data is negligible. As to the reduction of transactional checks, NoSQL stores provide better access speeds than relational databases. Bogdan et al. compared various databases and found that NoSQL databases provided better read and write latencies in a write intensive environment than relational databases [13]. Specifically Cassandra [14] is well established in high load environments [15] and was therefore chosen to be integrated in the framework. Another benefit of NoSQL data stores is the schemaless data model that allows the extension of the database schema at runtime. As mentioned before, collecting building related data at an urban level must be generic, easily extensible and flexible. To store generic data of various data sources, the need to refer to a datapoint was removed from the tuple definition. A generic data tuple is thus defined as $< tuple\_id, timestamp, value >$. The context of the measurment is established via a partition or node id and is not stored in the tuple itself. To develop a sustainable scaling strategy, the data store requirements were calculated as followed: To allow real-time data analysis a short recording interval is chosen. It is assumed that one entity contacts the database every second. This results in $3.15*10^7$ operations per year. The tuple size is calculated by considering the data type (64-bit IEEE-754 floating point) and the Cassandra specific overhead of 39 bytes per tuple. This results in 1.18 GB data per year for one sensor. Considering the Cassandra specific maximum file size (5 TB) specifies that one file can hold 4237 years of data per sensor. NoSQL stores distribute data randomly accross nodes. To increase read performance, 30-day shards are introduced. One monthly shard can hold $2.6*10^6$ measurements.

3 Application Architecture

3.1 Highly reliable and scalable Design

A scalable architecture is realized by a distributed system that is based on loosely coupled modules for

- data processing
- data retrieval
- data persistence
- data access
- data presentation

To reduce the database load that is induced by processing routines [9] the responsible logic was extracted from the database and moved to an indepedent module in a higher application layer. To realize modularity, scalability and to increase reliability on the application level, modules can be deployed redundantly and on different machines. Stateless core components allow new instances to be added during runtime, for instance to serve load peaks (i.e. monitoring occupancy during the morning rush hour) and to be removed in the cooling-down period.

Such a concept implementation requires a central distribution mechanism that routes requests between modules, respective physical machines that are distributed via a city's buildings. As the proposed framework is written in Java, all components are bundled by a Message Oriented Middleware (MOM) that is accessed via a Java Message Service (JMS) API.

The communication process is established by dynamically created queues (point-to-point) and topics (publish-subscribe). On binary protocol level, the Advanced Message Queuing Protocol (AMQP) was chosen, as it is secure, reliable, high performant and vendor-neutral [16].

Every single instance of redundantly deployed components listens to the same queue. The transferred request messages are always handled by exactly one instance (the one who initially took the message from the queue). Changes in datapoint values can be observed by subscribing to a datapoint's topic. That specific topic is identified via the owning datapoint's unique name and the prefix "*OBSRV_*": *OBSRV_<dp-name>*. For instance, the activity observation of the specific sensor "con1" (contact sensor) would be realized via the topic id *OBSRV_con1*.

3.2 Virtual Datapoint Collections

Datapoints always represent physical entities (for instance sensors). As has already been mentioned, the concept of virtual datapoints realizes data representations that are not bound to a physical sensor. For instance, this might be the on demand calculation of an entire building's accumulated energy consumption, average temperature or requesting values from a weather forecast. As VDPs work at a very low application level and behave like native datapoints, other application parts can use them transparently. VDPs can be seen as plug-in components

and are extensively used to support data simulation, model calibration and data prediction. The implementation of a virtual datapoint is deployed as a distinctive component and is accessed through a dynamically created queue which's name is derived from the VDP's type by prefixing "*DATA_*": *DATA_<type>*. For instance, a VDP that provides Radiance [17] simulations could use the queue *DATA_RADIANCE*. VDPs that implement the same type share the same queue which realizes support for multiple instances and hence increases reliability and availability.

VDPs offer a way to establish communication between buildings and components within one application context which is an essential step towards a distributed urban monitoring and simulation structure. However, VDPs can only return a primitive value, for instance the aforementioned overall energy consumption. To decrease configuration effort for buildings we introduce Virtual Datapoint Collections (VDC) that combine a set of VDPs to a bundle. This bundle is configured once and applies standardized calculations on the respective building's data. A VDC uses the same communication process and protocols as described in the VDP concept.

3.3 Security

Application access is only granted to registered users. By default, communication between components is entirely encrypted via SSL to prevent eavesdropping of sensitive data. Data access is handled at zonal level but will not be introduced in the scope of this paper.

4 Results

The application is divided into four layers. As Fig. 3 shows, these are a persistance, service and presentation layer, as well as a service adapter that offers access to the data collection via standardized protocol implementations (OPC Unified Architecture, oBIX, REST). The presentation layer consists of two client applications, a web client and a mobile client. The service layer implements the BMS business logic (data processing) and the VDCs. The persistence layer manages the relational master data store and the NoSQL datapoint value store. To allow a distributed module deployment, all components communicate via the MOM.

Fig. 4 illustrates the specific module implementations of the persistence layer after the revising process. Configuration and security sensitive data is stored in a MySQL database. Sensor configurations are partly kept in BIM models that are managed by a BiMserver instance. The metadata-module handles configuration requests and accesses the two databases. Datapoint values are either stored in the well tested initial MySQL environment, in a Cassandra cluster or a neo4j graph database. Neo4j support was partly introduced to monitor relationships between urban entities, partly to store sensor measurements. The configuration data and datapoint data is merged and connected in the persistence-module.

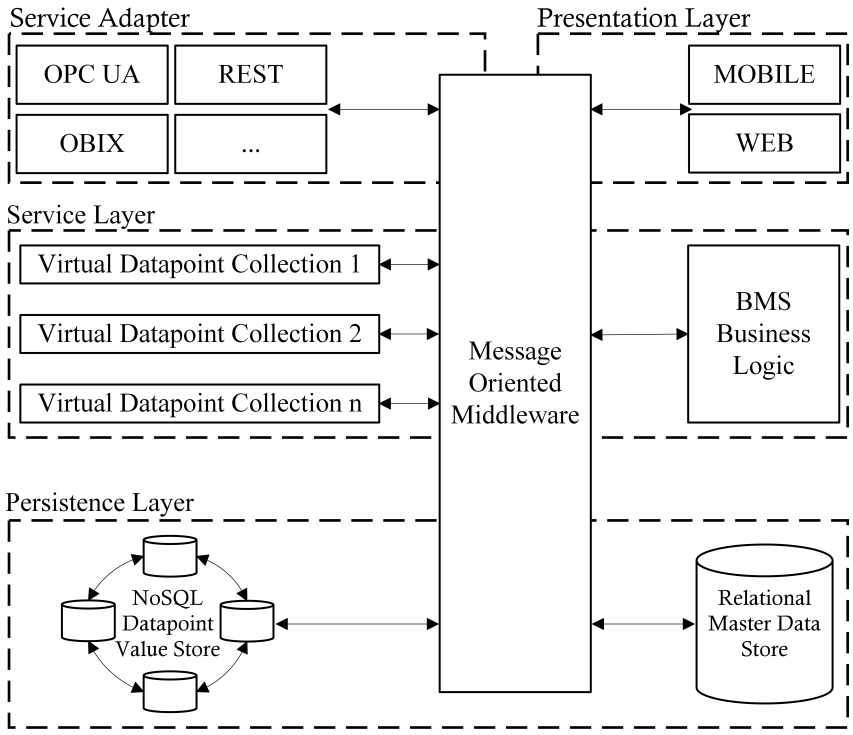

Fig. 3. The proposed urban monitoring framework's system architecture.

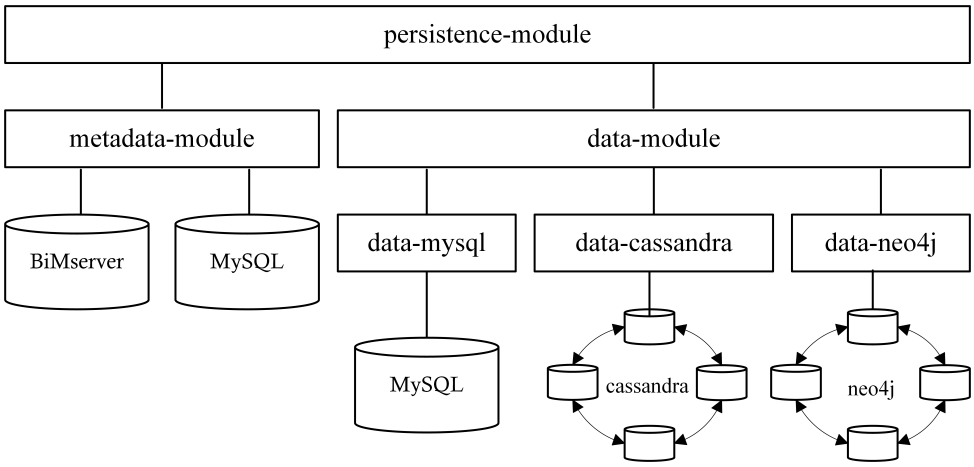

Fig. 4. Persistence layer: specific module implementation.

This module also implements the processing functionality that was extracted during the refactoring process.

5 Conclusion

The present work describes the efforts to scale a generic building monitoring system and the underlying data structure to support a distributed urban data exchange process. This process focuses on two main parts:

- building a highly generic data collection that offers an extensible and flexible data model
- developing a distributed application that offers standardized communication between physically distinct deployed modules

The ideas and concepts presented in this work should be understood as platform-independent and provide a first step towards the realization of an urban data warehouse. The developed hybrid data store that includes both relational and non-relational components will be tested against the well established relational base system in terms of both reliability and performance. This will be realized not only by running extensive database tests, but also by simultaneously deploying both systems in a project environment.

References

1. Mahdavi, A. From Building Physics to Urban Physics. In *Stadt: Gestalten*, chapter 34, pages 181–186. Springer Vienna (2012)
2. Schuss, M., Tahmasebi, F., Vazifeh, E., Mahdavi, A. The influence of online weather forecast uncertainties on the performance of predictive zone controllers. In Josef Hraska et al., editors, *enviBUILD 2013 - Buildings and Environment*, 1, 2013. STU - Nakladate#318;stvo STU, Bratislava (2013)
3. Saipi, N., Schuss, M., Pont, U., Mahdavi, A. Comparison of simulated and actual energy use of a hospital building in austria. In Josef Hraska et al., editors, *enviBUILD 2013 - Buildings and Environment*, 1, 2013. STU - Nakladate#318;stvo STU, Bratislava (2013)
4. Pang, X., Wetter, M., Bhattacharya, P., Haves, P. A framework for simulation-based real-time whole building performance assessment. *Building and Environment*, 54(0):100–108 (2012)
5. Zach, R., Glawischnig, S., Hönisch, M., Appel, R., Mahdavi, A. Most: An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization. In Gudni Gudnason, Raimar Scherer, et al., editors, *eWork and eBusiness in Architecture, Engineering and Construction*, pages 97–103. Taylor & Francis (2012)
6. Zach, R., Mahdavi, A. Most - designing a vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization. In EPOKA Univ.; Dep. of Arch., editor, *Proceedings - First International Conference on Architecture and Urban Design - 1-ICAUD*, 1, Epoka University Press (2012)

7. Wolosiuk, D., Ghiassi, G., Pont, U., Shayeganfar, F., Mahdavi, A. Mahdavi, Stefan Fenz, Johannes Heurix, Amin Anjomshoaa, and A Min Tjoa. Semergy: Performance-guided building design and refurbishment within a semantically augmented optimization environment. In Josef Hraska et al., editors, *enviBUILD 2013 - Buildings and Environment*, 1, 2013. STU - Nakladate#318;stvo STU, Bratislava (2013)
8. Zach, R. *An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization.* PhD thesis, Abteilung Bauphysik und Bauökologie, Institut für Architekturwissenschaften (2012)
9. Zach, R., Hofstätter, H., Glawischnig, S., Mahdavi, A. Incorporation of run-time simulation-powered virtual sensors in building monitoring systems. In IBPSA, editor, *Building Simulation 2013 - 13th International Conference of the International Building Performance Simulation Association.*, pages 2083–2089. IBPSA (2013)
10. Zach, R., Tahmasebi, F., Mahdavi, A. Simulation-powered virtual sensors in building monitoring systems. In Ardeshir Mahdavi and Bob Martens, editors, *Proceedings of the 2nd Central European Symposium on Building Physics 9-11 September 2013, Vienna, Austria*, pages 657–664, 1 (2013)
11. Zach, R., Schuss, M., Bräuer, R., Mahdavi, A. Improving building monitoring using a data preprocessing storage engine based on mysql. In Gudni Gudnason, Raimar Scherer, et al., editors, *eWork and eBusiness in Architecture, Engineering and Construction*, pages 151–157. Taylor & Francis (2012)
12. Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., Abramov, J. Security Issues in NoSQL Databases. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 541–547 (2011)
13. Tudorica, B. G., Bucur, C. A comparison between several NoSQL databases with comments and notes. In *Roedunet International Conference (RoEduNet), 2011 10th*, pages 1–5 (2011)
14. Cassandra. The apache cassandra project, January 2014.
15. Han, J., Haihong, E., Le, G., Du, J. Survey on NoSQL database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366 (2011)
16. Vinoski, S. Advanced message queuing protocol. *Internet Computing, IEEE*, 10(6):87–89 (2006)
17. Radiance. A validated lighting simulation tool, January 2014.