Proceedings of the 2nd ICAUD International Conference in Architecture and Urban Design
Epoka University, Tirana, Albania, 08-10 May 2014
Paper No. 193

Application of Web-Technologies in Building Information Systems: the case of the Monitoring System Toolkit (MOST)

Stefan Glawischnig, Harald Hofstätter and Ardeshir Mahdavi
Department of Building Physics and Building Ecology, Vienna University of Technology
Karlsplatz 13, 1040 Vienna, Austria stefan.glawischnig@tuwien.ac.at,
harald.e259.hofstätter@tuwien.ac.at, amahdavi@tuwien.ac.at

ABSTRACT

Building information systems collect building related sensor data. This data is processed and distributed to various users. The present contribution discusses the application of web technologies in building information systems and aims to give an overview of interesting applications and current research and development fields. The analysis is conducted using the example of the Monitoring System Toolkit, a vendor and platform independent building monitoring system that is developed at the Department of Building Physics and Building Ecology at the Vienna University of Technology. This work aims to provide a technological overview of communication processes, data storage and distribution mechanisms as well as representation workflows. The migration process from relational to non-relational data stores is introduced, including the extraction of processing logic from a relational database. Furthermore, the concepts behind homogenous data organization (data structures) and communication flows across the application layers and services are introduced.

KEYWORDS: Building Monitoring System, Scalability, Distribution, Performance Optimization

1 INTRODUCTION

Previous research and development efforts by the authors and other groups elaborated on the potential benefits that could be gained by the concurrent, real-time analysis of multiple building data streams. Within the scope of recent research efforts, a vendor and platform independent building-monitoring system (MOST 2012) was developed that provides a structure to store, communicate and analyse building sensor data within one application context. The development process follows a distributed, service-oriented approach. The currently available service implementations concern data integration, processing, storage and communication (Figure 1). Sensor data is passed to the data store via three possible methods: a connector implementation, the application's service layer or a virtual datapoint (Zach et al. 2013).

Datapoints always represent physical entities, for instance sensors. Virtual datapoints can be used to derive on demand information about physical phenomena, which are not directly monitored (for instance average room temperature for a certain period). Furthermore, virtual datapoints can be used to include simulation functionality into monitoring applications. In this case a virtual datapoint is a distinct component that is executed on demand by the application core and runs optimization or simulation tasks.

The connector service allows a vendor independent communication process with common building monitoring systems and automation technologies. Additionally, connectors can be used to integrate

historical datasets, for instance CSV weather data, into the data store. The columns in a CSV-file are mapped to database entities and stored as data tuples (Zach 2012). Beside connectors, data is accessible via two standard industry protocols, OPC Unified Architecture and Open Building Information Xchange (oBIX), and custom RPC (Remote Procedure Call) and REST (Representational State Transfer) implementations. The data storage service handles the databases and data sources.

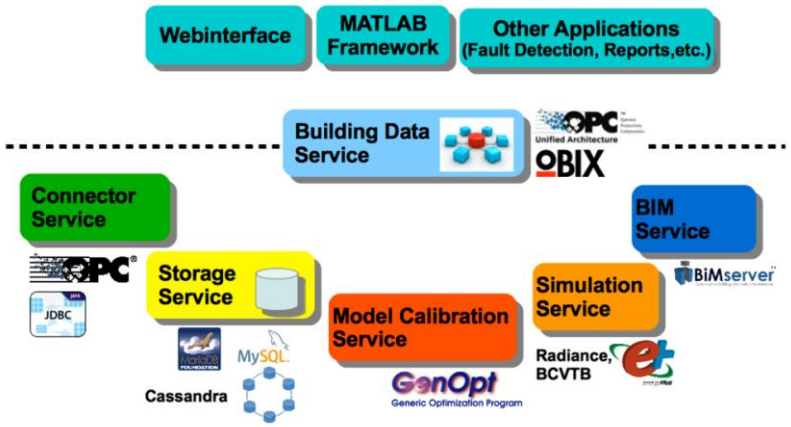

Figure 1: Monitoring system service architecture (Zach et al. 2013).

1.1 Approach

The present contribution introduces the most recent framework enhancements. On database level, new data storage mechanisms that intend to resolve performance bottlenecks were developed. Support for a distributed NoSQL data store and data structure was included to provide support for concurrent monitoring of multiple buildings. The relational database's processing functionality was translated from SQL to Java code and separated into distinct modules to improve code performance and maintenance. An object structure that enables homogenous data handling across the application core's boundaries was implemented.

2 METHODOLOGY

2.1 Data Store

Database performance problems raised the need to improve the data store mechanisms. Currently, the relational MySQL database handles 80.000 transactional commits, 130.000 inserts, 700.000 selects and 53.000 updates per day. Experience showed that a majority of the database induced network traffic (> 90%) concerns operations on the relational database's data table (Figure 2). Furthermore, the execution of data processing procedures (Zach et al. 2012) eventually made the database unresponsive. Specifically, the calculation of periodic data proved to be responsible for performance break downs. To resolve this issue a threefold approach was implemented: (i) Data processing routines were extracted from the database logic and moved to a distinct framework module. (ii) Where possible, data processing was outsourced to virtual datapoints. (iii) The data table was removed from the database schema and was moved to a NoSQL cluster.

A comparison of databases (Tudorica and Bucur 2011) showed that NoSQL databases offered better read and write latencies in write intensive environments than relational databases. Whilst NoSQL databases still functioned, MySQL got unresponsive at approximately 7000 read or write operations. A key-value store was chosen to hold the write intensive sensor measurements due to the structure of sensor measurements (value, timestamp and sensor reference id). Moreover, sensors submit small byte-

sequences on a periodic basis, which allows an accurate estimation of needed resources. Cassandra, initially developed by Facebook, is well established in high load environments (Jing et al. 2011) and offers a powerful query language, similar to SQL (Cassandra 2014). The feasibility of Cassandra as a value data store was assessed by the following simple calculation:

A sensor records a measurement every second. This corresponds to 3.15×10^7 operations per year. Considering the structure of a measurement (value, timestamp and sensor reference), the data type (64-bit IEEE-754 floating point) and the Cassandra specific overhead of 39 bytes per tuple, a magnitude of 1.18 GB data per year per sensor is resulted. Given the maximum Cassandra file size of 5TB, one node can therefore store one year of data for 4237 sensors. As a new physical server is added to the cluster, the database core initializes a new node and automatically synchronizes the new data source. This is possible due to the reduced ACID (Atomicity, Consistency, Isolation and Durability) property checks of non-relational data stores. Considering the nature and availability of building sensor measurements (intervals i=[1, 3600] sec), we can conclude that the effect of one lost measurement on an entire day's data collection is negligible.

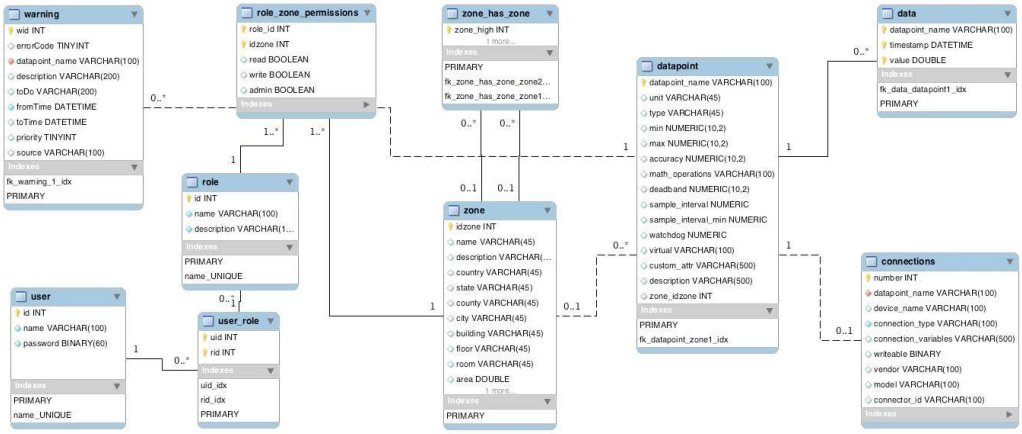

Figure 2: Relational database schema

2.2 Data processing routines

The initial relational data store (MySQL) offered data access functionality encapsulated in stored procedures. Table 1 gives an overview of these functions' coverage (read, write). The procedures' SQL-statements (Structured Query Language) were analysed and moved to a Java module that is deployable on a webserver and can be reused within the scope of other research applications. To generalize data exchange, data tuples are communicated through *DatapointDatasetVOs*. Datapoint-dataset value objects are serializable ArrayLists, containing data tuples and offering simple data access functionality (for instance, returning data before a certain timestamp).

The processing logic that was part of MySQL stored procedures (Table 1) was moved to logically organized Java classes. For instance, the *getValuesPeriodic()* procedure logic was separated into four classes (Figure 3): a *CalcAverageDataHolder, CalcAverageData* class that holds and organizes data, a *PeriodicDataGenerator* that calculates periodic data and returns *DatapointDatasetVOs* and a *PeriodicDataComparison* class that contains the logic to compare two *DatapointDatasetVOs* and is thus reusable in various application fields.

Table 1: Stored procedures that access and manipulate the relational schema's data table (based on Zach et al. 2012, extended)

Procedure name	Read	Write
addData(dp, ts, value)	x	x
addDataForced(dp, ts, value)		x
emptyDatapointTimeslot(dp, start, end)	x	x
calcAverageWeighted(dp, start, end, start value)	x	
emptyDatapoint(dp)	x	x
getNumberOfValues	x	
getValues(dp, start, end)	x	
getValuesPeriodic(dp, start, end, period, mode)	x	
getValuesPeriodicAnalog(dp, start, end, period, mode)	x	
getValuesPeriodicBinary(dp, start, end, period, mode)	x	
getValuesPeriodicWhereDpBetween(dp1, start, end, period, dp2, valueLow, valueHigh, modeDp1, modeDp2)	x	
getValuesPeriodicWhereDpBigger(dp1, start, end, period, dp2, value, modeDp1, modeDp2)	x	
getValuesPeriodicWhereDpEqual(dp1, start, end, period, dp2, value, modeDp1, modeDp2)	x	
getValuesWhereDpLower(dp1, start, end, period, dp2, value, modeDp1, modeDp2)	x	
getValuesWhereDpBetween(dp1, start, end, dp2, valueLow, valueHigh)	x	
getValuesWhereDpBigger(dp1, start, end, dp2, value)	x	
getValuesWhereDpEqual(dp1, start, end, dp2, value)	x	
getValuesWhereDpLower(dp1, start, end, dp2, value)	x	
interpolateValuesLinear(dp, start, end, period, value start, value end)	x	

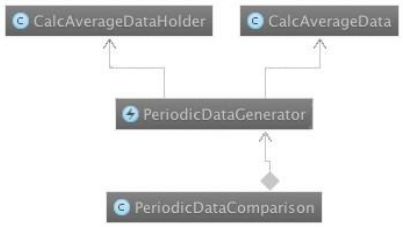

Figure 3: Refactored processing logic.

2.3 Communication

The introduced data value objects (for instance *DatapointDatasetVO)* reside in the core of the system. As they partly encapsulate business logic and are specifically designed to handle sensor measurements, value objects are not optimized to be transferred across the application-core boundaries. Data Transfer Objects (DTO) are introduced to communicate data outside the application core. So far, the following DTOs are implemented: *DpDataDTOs* represent single data measurement tuples; *DpDatasetDTOs* hold multiple *DpDataDTOs* and offer enhanced data access routines (e.g. get datasets before a certain timestamp). Figure 4 shows a very simplified communication layout of the application. The DTOs are transferred via the application layers to be used in various modules. These might be the simulation service as well as a virtual datapoint or the server package of the web application.

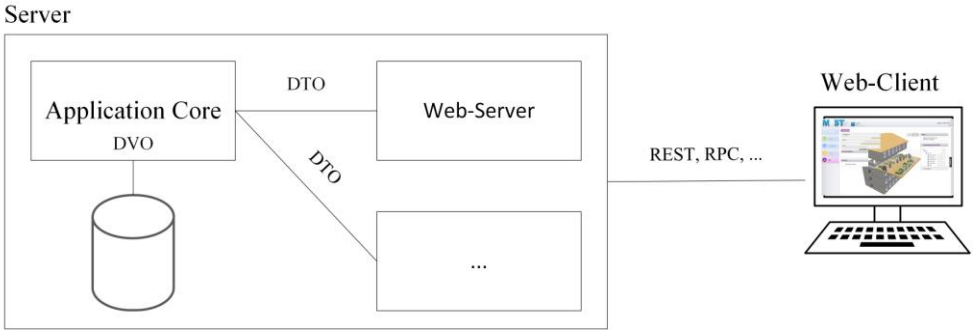

Figure 4: Simplified communication diagram, within the application context.

2.4 Virtual Datapoints

Virtual datapoints work at a relatively low application level and thus behave similar to native datapoints. Application parts that have access to the service layer can use them transparently. A virtual datapoint or virtual sensor is a distinct component that is accessible by the building monitoring's business logic through predefined interfaces. Beside the possibility to use virtual datapoints for periodic data calculation (Zach et al. 2013), the model calibration and simulation services apply the virtual sensor concept to include generic optimization (GenOpt 2014) and simulation (Radiance 2014, EnergyPlus 2014) into the toolkit. If an EnergyPlus simulation is requested on demand, the virtual datapoint requests the respective DTOs from the storage service and prepares and executes a call to the simulation program (Figure 5). The exact procedure is documented by Tauber et al. 2014.

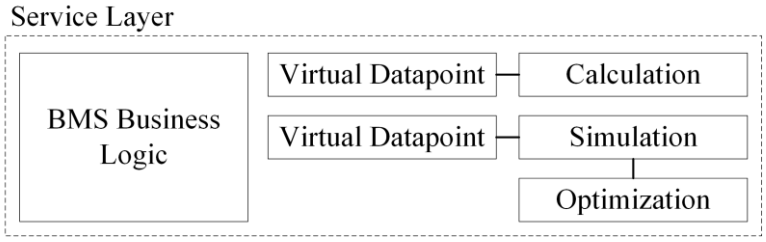

Figure 5: Service layer participants.

3 RESULTS

The refactoring process intended to generalize the application structure. The processing logic that was encapsulated in MySQL stored procedures was separated into three distinct modules: *Comparator*, *Validator* and *Generator* (Figure 7).

The *Comparator* module offers the functionality to compare generic datasets regarding equality and homogeneity. This might either be a simple comparison of data tuples based on logical operators, or a more complex comparison of periodic datasets, where data tuples should meet predefined conditions. Data validation concerns persistence integrity. Before a sensor measurement is added to the database it is tested against certain boundary parameters (for instance sample intervals, deadband, value ranges, etc). Based on the sensor definitions, the result of the persistence validation determines whether a measurement is discarded (e.g. too small sample intervals) or a warning is generated (possible sensor fault). The periodic data calculation algorithms that were encapsulated in the database logic (Table 1) were moved to the *Generator* module. Each method receives and returns standardized datapoint datasets. From a structural point of view, periodic datasets are treated similar to native datasets. This allows standardized and reusable workflows and improves code maintenance. The *preproc-library* is included in

the respective data storage module that is used by the specific monitoring framework implementation (for instance *most-data-cassandra*, Figure 7). This data store module executes database queries and routes datasets between the database and the application core. To offer support for various databases and to handle the respective implementations, a generic *data* module was implemented. As previously mentioned, only standardized datasets are used to communicate data throughout the application context. The *data* module offers an interface to map specific database implementations to the generic datasets (Figure 6). This way it is possible to include different data stores for separately deployed application instances. Currently, support for MySQL and Cassandra is provided.

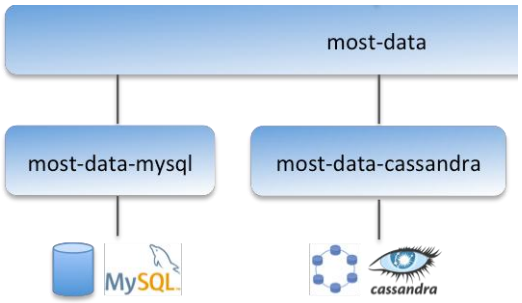

Figure 6: Modules responsible for data handling (Glawischnig et al. 2014, modified)

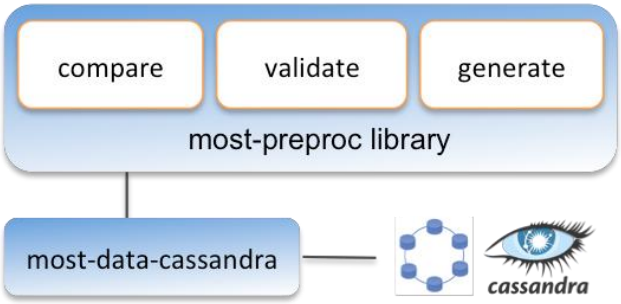

Figure 7: preproc-library contents (Glawischnig et al. 2014).

4 CONCLUSION

The presented technologies are extensively used in web development. For instance, Cassandra was initially developed by Facebook before it was integrated into the Apache open source program. Currently, support for neo4j (Neo4j 2014) is integrated, a graph database that is used by social networks to build relationship graphs. The work presented in this paper provides an overview of recent efforts to build a generic, extensible and modular building monitoring system framework. Future work will focus on the integration of distributed working software, specifically Apache Hadoop (Hadoop 2014) to improve data processing routines. Currently, these routines (e.g. periodic data generation) do not work in a distributed manner, which can result in performance bottlenecks. The migration from solely relational data stores to non-relational datastores offers an efficient (performance and economic) way to organize multiple buildings' data in a singular system. Furthermore, distributed data stores improve scalability and simplify the possible application of cloud stores.

REFERENCES

Cassandra 2014. The Apache Cassandra Project. http://cassandra.apache.org. last visited: February 2014

EnergyPlus. 2014. Building Technologies Office: EnergyPlus Energy Simulation Software. http://apps1.eere.energy.gov/buildings/energyplus/. Last visited: February 2014.

Genopt. 2014. GenOpt-Generic Optimization Program. http://www.simulationresearch.lbl.gov/GO/. Last visited: February 2014.

Glawischnig, S., Hofstätter, H., Bräuder, R., Zach, R., Mahdavi, A. 2014. Bridging RDBMS and NoSQL to build a high-performance and scalable storage engine for building information systems. Accepted for publication. ECPPM 2014, Vienna, Austria.

Hadoop. 2014. Apache Hadoop. http://hadoop.apache.org. Last visited: February 2014.

Jing, H., Haihong, E., Guan, L., Jian, D. 2011. Survey on NoSQL Database. ICPCA 2011, Port Elizabeth, South Africa, ISBN: 978-1-4577-0209-9. 2011, pp. 363-366

MOST. 2012. Monitoring System Toolkit. http://most.bpi.tuwien.ac.at. Last visited: February 2014.

Neo4j. 2014. Neo4j – The world's leading Graph Database. http://www.neo4j.org. last visited: February 2014.

Radiance. 2014. Radiance-A validated Lighting Simulation Tool. http://www.radiance-online.org. Last visited: February 2014.

Tauber, C., Tahmasebi, F., Mahdavi, A. 2014. Automated simulation model calibration based on runtime building monitoring. Accepted for publication. ECPPM 2014, Vienna, Austria.

Tudorica, B.G. , Bucur, C. 2011. A comparison between several NoSQL databases with comments and notes. Proc. ProEduNet 2011, Iasi, Romania, ISBN: 978-1-4577-1233-3, 2011, pp.1-5.

Zach, R. 2012. An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization. PhD thesis, Departement for Building Physics and Building Ecology.

Zach R., Schuss M., Bräuer R., Mahdavi A. 2012. Improving building monitoring using a data preprocessing storage engine based on MySQL. In "eWork and eBusiness in Architecture, Engineering and Construction". Tayler & Francis, ISBN: 978-0-415-62128-1. 25 – 27 July, Reykjavik, Island. pp. 151-157.

Zach, R., Tahmasebi, F., Mahdavi, A. 2013. Simulation-powered virtual sensors in building information systems. In "Proceddings of the 2nd Central European Symposium of Building Physics" 9-11. September 2013, Vienna, Austria, pp. 657-664.