TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DISSERTATION

AN OPEN-SOURCE, VENDOR AND TECHNOLOGY INDEPENDENT TOOLKIT FOR BUILDING MONITORING, DATA PREPROCESSING, AND VISUALIZATION

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von
Univ.-Prof. Dipl.-Ing. Dr.techn. Ardeshir Mahdavi
E 259-3 Abteilung für Bauphysik und Bauökologie
Institut für Architekturwissenschaften

eingereicht an der
Technischen Universität Wien
Fakultät für Architektur und Raumplanung

von
Mag. DI(FH) Robert Zach
Matr. Nr.: 0726404
Gebharts 46, 3943-Schrems

Wien, im Juli 2012

Zusammenfassung

Der Energieverbrauch von Gebäuden entspricht in den meisten Ländern einem signifikanten Teil des Gesamtenergieverbrauches. Um Gebäude optimal betreiben zu können, sind Informationen über aktuelle Energieverbraucher, Anforderungen der Gebäudebenutzer und aktuelle Steuerungsbefehle von wesentlicher Relevanz. Allerdings sind passende Sensorik und Datenverarbeitungsstrategien in der überwiegenden Mehrzahl bestehender Gebäude nicht vorhanden. Aber auch Neubauten vermissen meist Möglichkeiten benötigte Daten in passender Form weiterzuverarbeiten.

Diese Arbeit beschreibt mögliche Datenerfassungs-Infrasstrukturen um unterschiedliche Gebäudedaten (Energieverbräuche, Komfortparameter, usw.) in Echtzeit zu verarbeiten. Solche Daten können natürlich für unterschiedliche Zwecke weiterverarbeitet werden. Zu diesem Zweck wird ein Toolkit, bestehend aus fünf Komponenten (Connector, Database, Data-Abstraction Framework, MATLAB Framework, Visualization Framework), vorgestellt. Das Toolkit ermöglicht die Erfassung und Verarbeitung der Gebäudedaten in unterschiedlichen Anwendungsgebieten. Es verfügt über leistungsfähige Datenaufbereitungs-Funktionen (z.b. generierung zeitlich strukturierter Datensätze), bietet Schnittstellen für eine automatisierte Stapelverarbeitung (MySQL, OPC-UA, usw.) und beinhaltet Anwendungen für Daten-Aggregation, Visualisierung und Analyse (z.b. Erstellung eines Mollier-Diagramm, Unterstützung unterschiedlicher Dateiformate, usw.).

Um den Nutzen des Toolkits in einem praktischem Kontext zu untersuchen, wurden zwei unterschiedliche Gebäude in Wien, Österreich als Referenzimplementierung gewählt. Eines der Gebäude ist ein Neubau und enthält bereits teilweise Infrastruktur zur Gebäudedatenerfassung. Die Arbeit zeigt wie das Toolkit verwendet werden kann, um Daten aus allen Gebäudeautomatisierungssystemen zu erfassen. Das zweite Gebäude ist ein Altbau und bietet keine wiederverwendbare Infrastruktur. Daher wird ein unabhängiges System zur Datenerfassung installiert.

Summary

In most countries the energy consumption of buildings represents a significant part of the overall energy usage. To run buildings with optimal operation regimes, information about actual energy consumers, user requirements and current building control actions is critical. However, the presence of appropriate sensing, data-storage, and data-processing capabilities is almost entirely absent in the vast majority of existing buildings. Even new buildings mostly miss the possibility to process and analyze building data in an appropriate way.

The present contribution describes possible monitoring infrastructures to collect and process building data (energy use, comfort parameter, etc.) in real-time. Such data can be, of course, applied for various purposes. Toward this end, a toolkit, consisting of five components (Connector, Database, Data-Abstraction Framework, MATLAB Framework, Visualization Framework), is presented. The toolkit facilitates beneficial use of building data in various processing applications. It provides powerful preprocessing functions (e.g., generation of temporally structured data sets), offers interfaces for batch processing (MySQL, OPC-UA, etc.) and includes applications for data aggregation, display, visualization, and analysis (e.g. psychrometric and thermal comfort chart plots, data encapsulation and export, etc.).

To probe and gauge the utility of the toolkit in a realistic and practical context, two distinct buildings in Vienna, Austria were used for reference implementation. One is a new building, with some existing monitoring infrastructure elements. The dissertation illustrates how this building's data can be accessed using the toolkit, by connecting to all kinds of building automation infrastructures. The second building is more than hundred years old and provides no reusable building automation infrastructure. Therefore, an independent system needed to be installed for monitoring.

Acknowledgments

The research presented in this dissertation is supported by funds from the program “Innovative Projekte” of the Vienna University of Technology, the Austrian Science Foundation (FWF): Project: I 545-N22 (Ubiquitous dynamic building performance monitoring) and the program “Neue Energien 2020” within the “Klima- und Energiefonds”. Additional support was provided by the division "Gebäude und Technik" (Amtsdir. Hodcek), which supplied us with real-world test beds. Moreover, the thematic link to the CAMPUS 21 project (Control & Automation Management of Buildings & Public Spaces in the 21st Century, no: 285729) provided further impulses for the realization of the research objectives.

Parts of the text in this thesis are adopted from papers, written in relation to the projects, and co-authored with Prof. Ardeshir Mahdavi and project-team members.

First of all, I would like to thank Prof. Ardeshir Mahdavi for his support, guidance and advising throughout my thesis. I highly appreciate the time he invested into me by giving suggestions and review my work.

Special thanks belong to all MOST team members and my colleagues at the Department of Building Physics and Ecology. I really enjoy the time here.

Finally, I would like to thank my mum who cooked for me so often.

Contents

Chapter 1.
Introduction

Given the enormous global challenges associated with accelerating resource depletion, energy crisis, and climate change (IPCC 2009), multiple serious efforts are needed to curtail energy consumption, waste of resources, and environmental pollutions. Thereby, the building sector plays a significant role. In most countries the energy consumption due to the building sectors represents 40% and more of the overall energy usage (Graubner und Hüske 2003).

To achieve optimal operational regimes, however, the availability of information on building's energy and environmental performance is critical. Collection of such information requires, however, the presence of appropriate sensing, metering, data-transmission, data-storage, and data-mining capabilities. Such capabilities are almost entirely absent in the vast majority of existing buildings (Neumann and Jacob 2008, Mahdavi et al. 2008). Even in newer – so-called intelligent – buildings, the incorporated infrastructures for zonally differentiated sub-metering and environmental sensing are often ineffectual: data is not monitored in a comprehensive and consistent manner, multiple streams are not integrated, monitored data is not strategically stored, actual operational routines seldom make use of the potential of whatever data that is monitored and stored. In this context, substantial research and development efforts are necessary toward realization of highly versatile and scalable systems for concurrent and multi-layered energy, performance, and occupancy data acquisition and operational optimization in buildings.

Moreover, to realize a higher level of energy and resource efficiency in the whole buildings sector, the consideration of the existing building stock is crucial. For example, in Austria, new construction volume per annum represents less than 1% of the existing building stock's volume (Statistik Austria 2007). The factual energy conservation potential of new buildings

is even less than what this figure suggests, as new buildings are, in comparison to the existing ones, more energy efficient. New buildings offer of course a field of opportunities to conceive, develop, implement, and test new energy-efficient products and technologies. But the existing building stock represents undoubtedly the true challenge for improving the sustainability of the built environment. Toward this end, object-centered hardware developments (e.g. better thermal insulation of the building envelope, incorporation of efficient mechanical and electrical equipment, passive and active solar energy methods and devices) are important, but not sufficient. Specifically in the context of the existing building stock the intelligent (energetically and resourcially optimized) building operation is of utmost importance.

1.1. Motivation

Currently available building management systems (BMS) could be improved in view of a number potentially important functionality. For example, real-time data access to salient – and often dynamically changing – building information (such as zone temperature and energy use) could be provided via appropriate data processing and performance modeling applications (such as spreadsheets, mathematical routines, simulation tools). Moreover, additional data processing functionalities could be offered, involving, for example, the calculation and display of performance data in a structured spatial and temporal manner. These improvements would facilitate the exploitation of the critical benefits that could result from the integrated and concurrent analysis of multiple building data streams (Raftery et al. 2010, O'Donnell 2009). Such benefits include:

- Operation energy optimization through improved management of technical building systems.
- Increased awareness in building users regarding their impact on buildings' energy use.

- Early detection (and treatment) of deficiencies and malfunctions in energy systems and devices, thus effectively supporting a preventive maintenance regime.
- Successive building performance improvement and optimization via the analyses of dynamically updated building energy and performance databases.
- Long-term accumulation of empirical information on buildings' energy and environmental performance toward improving the design, construction, and operation of existing and new buildings.

To facilitate data utilization for all interest groups, open software interfaces play an important role. Technical limitations (no real-time data access, missing interfaces for batch processing, etc.) can prevent processing applications from accessing building data in the required way. Unstructured measurement data sets can further increase analyses effort and reduce exploitation potential. Therefore, application independent interfaces and appropriate data preprocessing (calculation of timely structured data sets, linked queries of energy use for specific time intervals and building zones under specific occupancy conditions, etc.) is critical to maximize data usage for all interest groups (facility manager, occupant, building owner, etc.).

It can be argued that, due to the long life cycle of buildings and the fast evolution of building technologies, a vendor and technology independent approach for monitoring is essential to ensure future-proof comprehensive data collection. Furthermore, independence from the building market developments can be increased by using open-source technologies.

1.2. Research statement

The following postulates the starting point of the research:

A) Building monitoring without appropriate data preprocessing reduces significantly the exploitation potential.

Building performance optimization based on building monitoring is a common approach (Gökce 2010). Many use cases can be expected from the collected data. However, possible exploitation potential is often not sufficiently covered. One reason is that data analyses effort can rise significantly if data is not available in the required structure.

B) Vendor and technology independent building monitoring is of central importance for future proof domain comprehensive data collection.

Due to the long life cycle of buildings and the fast evolution of building technologies, a vendor and technology independent monitoring approach reduces the risks of vendor lock. Using open-source technologies further increases independents of market development (Cassia 2007).

C) Open software interfaces are necessary to facilitate data utilization for all interest groups.

Technical limitations (no real-time data access, missing interfaces for batch processing, etc.) can prevent processing applications from accessing building data in the required way. This can significantly reduce data usage for different interest groups.

D) Runtime building monitoring is required to ensure energy-efficient operation.

Many case studies show that building performance can be very different then planned, if not all important parameters are taken into correct account (Raftery et al. 2010, Mahdavi 2009). Since various parameters

(e.g. user behavior, system faults, etc.) can change over time, a continuous evaluation is required.

E) Support of multiple fieldbus and building management system technologies is essential for a generic building monitoring system.

Since different physical conditions often require distinct measurement approaches, requirements for the monitoring system can vary. Therefore, the best fitting measurement, data transfer and storage technology can be very different. Given that no universal fieldbus and building management systems have emerged yet, a future proof building monitoring systems needs to support various technologies.

1.3. Structure

This dissertation is structured in terms of seven sections. Section 2 provides general information regarding the research approach and the requirements for the proposed building monitoring system. Section 3 describes each component of the developed monitoring toolkit in detail. Section 4 shows two real world implementations. Finally, section 5 discusses the research conclusions and future outlook. Section 6 lists references, figures and tables while section 7 contains documentation about various implementation details (flow diagrams, class diagrams, etc.).

Chapter 2.
Approach

To achieve the desired building monitoring characteristics, the Monitoring System Toolkit - MOST - is developed. Based on five components (Connectors, Database, Data-Abstraction Java Framework, MATLAB-Framework, Web-Visualization Framework), the toolkit facilitates beneficial use of building data in various processing applications. It provides powerful preprocessing algorithms (e.g., generation of temporally structured data sets), offers interfaces for batch processing (MySQL, OPC-UA, etc.) and includes applications for data aggregation, display, visualization, and analysis (e.g. psychrometric and thermal comfort chart plots, data encapsulation and export, etc.).

To evaluate real-life application scenarios, two buildings (housing a number of offices, labs, and lecture rooms of the Vienna University of Technology) were partly equipped with necessary monitoring infrastructures. One of the buildings is completed 2010 and provides – to a certain degree – a reusable building automation infrastructure. The second building was built more than 100 years ago and provides no reusable building automation infrastructure. As such, these buildings are representative of a large fraction of buildings in the existing building stocks. They represent a wide range of technical challenges that need to be met in order to realize the postulated dynamic data acquisition and processing architecture in the context of existing buildings. Such challenges pertain specifically to the technology update requirements for incorporation of high-resolution sensory and metering capabilities, device connectivity, and cross-platform data transfer. The described monitoring infrastructure provides a flexible base to handle these challenges and presents data processing applications to improve building performance.

2.1. Requirement analyses

To obtain the requirements for the proposed monitoring toolkit, essential building technologies are analyzed. Building communication networks are usually described via the three layer model defined in ISO 2004. This model is appropriate to describe network communication strategies, but lacks the coverage of sensor/actor technologies. It does not deal with the challenge of getting the information of different physical domains into an electrical signal and their different requirements regarding fieldbus networks. Since the proposed system focuses on the measurement and processing of different physical parameters, an additional Layer – Physical level – was added as shown in Figure 1. This Layer covers the technologies required to converts the desired data stream (e.g. temperature, relative humidity, etc.) into the digital domain.

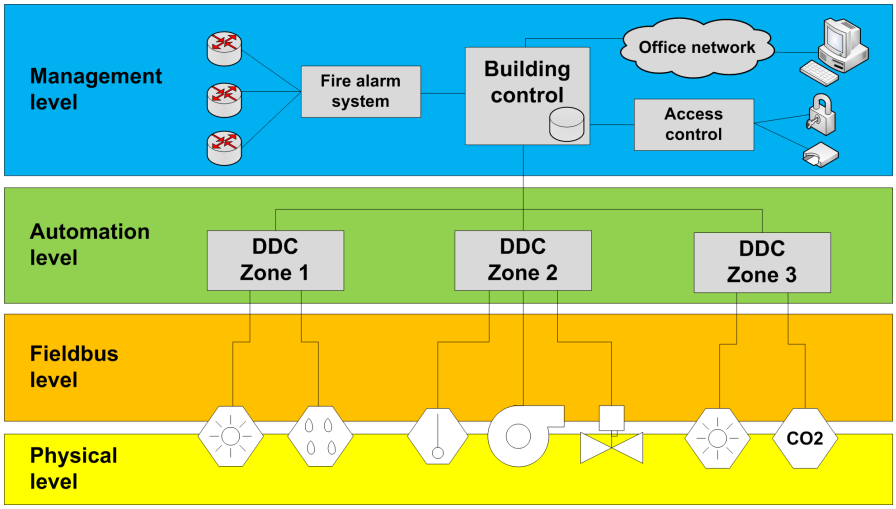


Figure 1. Four layer model of a generic monitoring system

2.1.1. Physical level

The physical level addresses pertinent sensory devices and technologies that are required toward an efficient, dynamic, and scalable acquisition of

the required data. Table 1 provides an overview of relevant data streams together with their required sensor technologies.

Table 1. Data streams and required sensor technologies

	Data Stream	Sensor Technology
i	Energy use	(Sub-)Meters for electricity, gas, oil, water, etc.
ii	Indoor environment	Temperature, humidity, CO2, VOC, illuminance, etc.
iii	Outdoor environment	Temperature, wind, rain, solar radiation, etc.
iv	Occupants' presence, actions, and feedback	Motion/presence detectors, number of people, location sensing, etc.
v	Environmental control systems states	Window and door states, blinds, etc.

Different sensor technologies do often require different fieldbus characteristics. For example, a presence-detector to turn on the light needs to send its information in a fraction of seconds while a CO2 sensor for heating, ventilation, and air-conditioning (HVAC) can work with longer intervals (i.e., minutes).

2.1.2. Fieldbus level

In a monitoring system the function of the fieldbus level is to transfer the measured data streams to the automation layer, which then acts as a backbone. Common service parameters for networks are throughput, reliability, security, scope, real-time, and power use. These characteristics can be used to describe a fieldbus, but it is not always possible to directly compare them based on these properties. For example, KNX has a much smaller throughput than LonTalk on the wire. But, depending on the system design and the grouping of devices in different KNX lines, the overall network load can be less than in a comparable LonTalk system. To get a starting point for technology decisions, current wired and wireless fieldbuses and their field of applications are listed in Table 2 and Table 3 (Daniels 2003, KNX 2004, Kastner et al. 2005, LON 2010).

Different monitoring strategies can be used to get the measured data from the sensor to the management level. The sensor can send measurements event triggered when some predefined conditions occur, or periodically with a fixed interval, or the Direct Digital Controller (DDC) can poll the stations. Depending on the data stream, different strategies fit best. For example, data from a presence detector is most accurate when an event-based strategy is used while an electrical meter is usually polled with a periodic interval to obtain a temporal view. An event-based strategy can also be used to reduce power consumption of battery or self-powered devices.

Table 2. Wireless fieldbus technologies and their field of applications

Fieldbus	**Field of applications**
ZigBee	Supports the dynamic creation of meshed networks which increases reliability and scope. ZigBee is based on the IEEE 802.15.4 standard and can work in the 2,4 GHz and 868 MHz ISM band. Most devices use the 2,4 GHz band which is often crowded when, for example, wireless local area networks (WLAN) are used too. ZigBee is used as a general purpose fieldbus.
EnOcean	Is optimized for low power consumption, which therefore allows the construction of self-powered sensor/actor devices. EnOcean uses the 868 MHz ISM band with amplitude modulation optimized for short packet transmission time and low power consumption. This increases throughput and reliability. EnOcean is mainly used for self-powered sensors and simple actuators.
KNX/RF	Is the wireless version of KNX. Only a few devices are available on the market at the time.
Z-Wave	Is designed for small systems in the field of home automation.
M-Bus/RF	Is the wireless version of M-Bus. Only a few devices are available on the market at the time.
IEEE 802.15.4	Defines only the lowest levels of a wireless communication. IEEE 802.15.4 is reused in some other standards and proprietary systems.

Table 3. Wired fieldbus technologies and their field of applications

Fieldbus	Field of applications
KNX/TP	General-purpose fieldbus. Used for lights, blinds and HVAC systems. KNX is the successor of the European Installation Bus (EIB) and is therefore mostly used in the European Union.
LonTalk	General-purpose fieldbus. Used for lights, blinds and HVAC systems.
M-Bus	Used for metering devices (electrical meter, heat meter, flow meter, etc.).
DALI	Used for controlling lights in isolated applications.

2.1.3. Automation level

The target of the automation layer in a monitoring system is to transfer all data streams to a central control station. It therefore acts as a backbone, which needs to handle higher data rates then the fieldbus networks.

Common network technologies in the automation level are Ethernet/IP, BACnet, KNX and LonTalk. Ethernet/IP provides high bandwidth, cheap mass-components and flexible integration possibilities and is therefore the most common technology for backbone networks. Underlining fieldbus packets can be encapsulated in Ethernet or IP frames (e.g. BACnet/Ethernet, BACnet/IP, KNX/IP, etc.) or the measured data can be directly transferred with pure Ethernet/IP communication (OPC Unified Architecture, proprietary Ethernet/IP protocol, etc.). Because Ethernet/IP provides only limited support for real-time data transfer and bandwidth allocation, a combination with an unpredictable office network can be problematic.

Pure BACnet, KNX or LonTalk provides only limited bandwidth, which is usually not sufficient for monitoring systems in the automation level.

2.1.4. Management level

The management layer handles the historical data storage, the visualization and the further processing of the data streams. Common technologies for historical storage and abstract data representation are

OPC (Data Access - DA, Historical Data Access - HDA, Unified Architecture - UA), BACnet/Web-Services (WS), oBIX and custom database designs.

OPC DA is highly used to provide a common software interface to different automation and fieldbus networks in the management layer. So called OPC DA servers abstract the sensors and actors as datapoints. The data of the OPC servers can then be accessed with OPC clients, which can be a user interface or any other processing application. OPC DA server provide only live data and run only on windows operating systems due to dependencies to the Distributed Component Object Model (DCOM 2012). To provide historical data access the OPC HDA standard or a custom database are popular solutions (Iwanitz and Lange 2002, OPC 2012).

To overcome the restriction of running OPC DA and OPC HDA server on windows only and to integrate all OPC sub-standards (DA, HDA, etc.) to a uniform technology, the OPC UA standard was created. It provides high potential, but is not fully supported by common products yet (Mahnke et al. 2009).

The standards BACnet/WS (ASHRAE 2004) and oBIX (OASIS 2006) provide comparable functionality as the OPC standards, but are only rarely supported by available products at the time.

2.2. Use cases

Use cases are investigated to obtain desired functionality for a general building monitoring system. Different interest groups and stakeholders presuppose diverse preferences and attitudes for building monitoring. To gain data regarding their attitudes and preferences, a user survey with 134 participants (occupants, guests, system developers, building operators, designers and facility managers) was conducted. The use cases listed in Table 4 are defined based on the results of the questionnaires (Chien et al. 2011).

Table 4. Exemplary use cases of the proposed visualization framework

Use case category	Use case examples
Visualize real time and historical data	- Show all temperature values in the zone X - Show the electrical energy consumption in the zone X from 01/2012 to 04/2012 - Show all zones, which have a temperature above 24 degrees - Show zones of dissatisfied users, the type and value of the reason (the reason is reported by the user) - Show the top 10 energy consumer zones (energy by square meter) of non occupied zones
Show prediction of future energy needs/costs	- What are the energy needs/costs for the next few days (based on the weather forecast, etc.) - On which days are which zones usually not occupied (e.g. by analyzing historical data. Most users are on holiday in calendar week x or on days x and y)
Give suggestions of what actions could optimize building performance	- Turn off the light and reduce the temperature in zones X - Suggest window and door states (to reduce overheating - natural cooling) - Suggest window and door states for the best cross-ventilation with current or predicted outdoor weather conditions - Suggest working times and days based on predicted work area conditions - Suggest work place for mobile workers based on their requirements and the building performance
Maintanance/Fault detection	- List defect sensors and actors - Show maintenance work needed right now - Show predicted maintenance work (e.g. change light x at y based on current usage pattern) - Early detection (and treatment) of deficiencies and malfunctions in energy systems and devices (by analyzing collected historical data from the building)

Chapter 3.

Monitoring System Toolkit

Based on the investigated requirements and use cases for building monitoring, the Monitoring Systems Toolkit – MOST – was developed. It focuses on vendor and technology independent building monitoring, data preprocessing, and visualization. Powerful data preprocessing algorithms and different software interfaces allow various applications to process desired building data streams. MOST also includes applications for data aggregation, display, visualization and analyses to simplify beneficial use of building data.

All software components of the proposed toolkit are shown in Figure 2. The toolkit provides *Connectors* to collect data from diverse building systems, a *Database* for historical data storage and data preprocessing, a *Data-Abstraction Framework* which serves different software interfaces, a *Webinterface* for simple data access and visualization, and a *MATLAB Framework* for complex data processing.

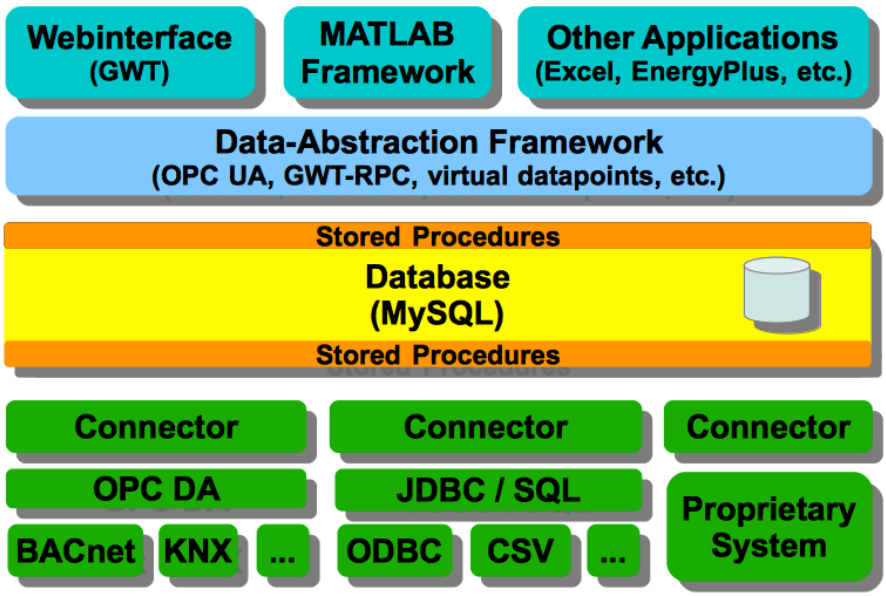


Figure 2. Software components of the Monitoring System Toolkit - MOST

All components can be used independently and are licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License (License 2012).

3.1. Connector

To support vendor independent communication with common building management systems and building automation technologies, diverse connectors were developed.

3.1.1. Connector framework

To simplify connector development, the following framework was implemented using the programming language Java. It abstracts each desired measurement point as a datapoint (stored in the MOST database) and each device driver (which reads the measurements from the respective sensor) as a connector object. A cutout of the class diagram is shown in Figure 3. The proposed connector software can be executed on any computer (or embedded system), which supports Java and provides access to the respective building sensor network and the desired monitoring server.

The startup method implemented in the class *ConnectorController* connects to the MOST database, loads required data source information and instantiates appropriate *Connector* objects. Each *Connector* object handles the communication between one sensor/actor and an associated datapoint of the MOST toolkit. It holds the following information:

- *dpName* – Defines the associated datapoint
- *connectionType* – Defines the type of connection which is supported by this connector (e.g. jdbc, opc-da, etc.)
- *connectionVariables* – Contains variables which are required for the respective connector. Each connector can store multiple variables separated with the ";" character within this argument. The syntax is defined as <variable name> <space> <value> <;>. For

example, to define the timezone of a data source and a desired polling interval, the variables *timezone* and *pollInterval* can be set using the following string "timezone Europe/Vienna; pollInterval 3600;". The respective connector can use the method *getVariable(String)* to read the value of defined variables.

- *writeable* – This flag tells if this connector is writeable. If the flag is set, the method *writeData(value)* is called for each value change of the associated datapoint. Write event propagation is implemented using the observer design pattern.
- *vendor/model* – The *vendor* and *model* parameter can be used to define device specific connector implementations.

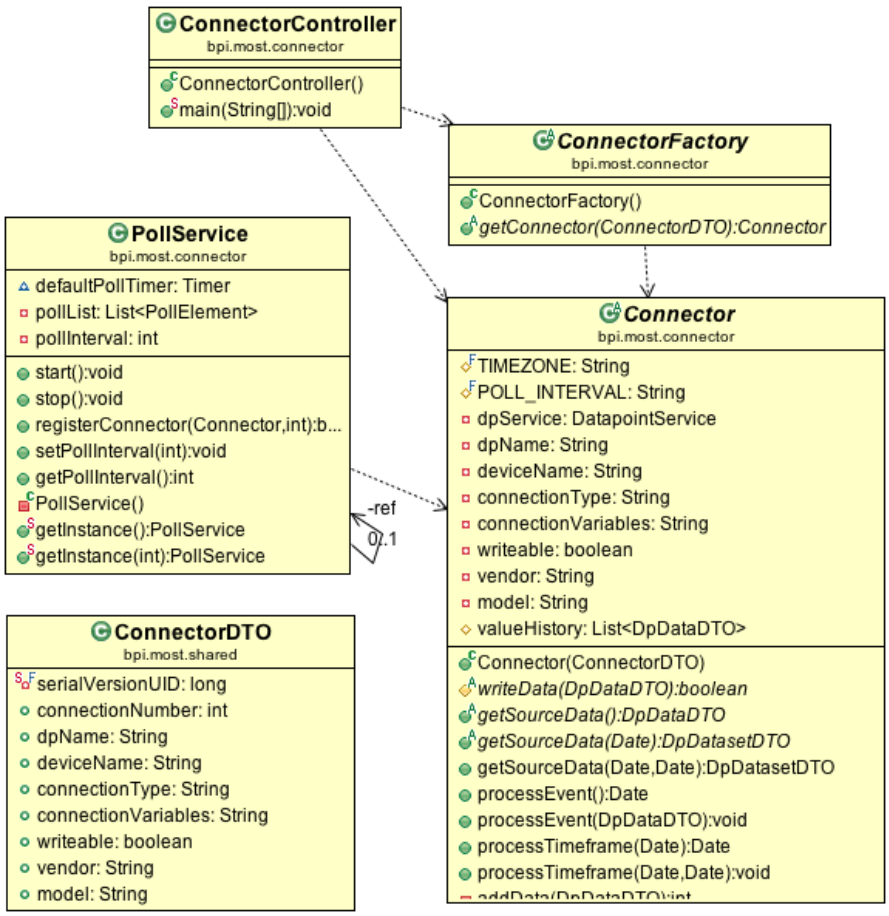


Figure 3. Class diagram of the connector framework

A datapoint can be connected to one or more devices (sensors/actors). Based on the requested data source information (connection type, connection variables, writeable flag, vendor and model - *ConnectorDTO*), appropriate *Connector* objects are instantiated. Different devices and communication technologies need different implementations. To enable the user to add support for additional devices during the building lifecycle, the Java Service Loader is used. The Java Service Loader searches for all classes, that implement the abstract class *ConnectorFactory*, instantiates it and calls the method *getConnector(ConnectorDTO dataSourceMetadata)*. Each *ConnectorFactory* implementation needs to check if the requested data source is supported. In this case, an appropriate *Connector* instance must be returned. This strategy enables the toolkit to add support for additional devices without recompiling the overall project (plugin concept). Generic connector implementations for diverse building related communication technologies (JDBC, OPC-DA, BACnet/IP, KNXnet/IP, etc.) are able to support standard conform devices. To add support for new or proprietary technologies/devices, the abstract classes *ConnectorFactory* and *Connector* must be implemented.

To register a new connector to the Java Service Loader, the fully qualified name of a class implementing the *ConnectorFactory* must be added to the file META-INF/services/bpi.most.connector.ConnectorFactory. Furthermore, the abstract class *Connector* needs to be extended with the actual driver implementation. Afterwards, a Java Archive (JAR) file must be generated and copied within the Java Classpath.

Abstract connector class

The abstract class *Connector* provides diverse generic connector functionality and defines methods that need to be implemented by intended connectors. For example, support for the connector variables *timezone* and *pollInterval* is implemented within this class. The variable *timezone* can be used to adapt the time offset of the sensor measurements (e.g. GMT+1) to the MOST database (UTC). For example,

by setting the variable to the timezone of Austria ("timezone Europe/Vienna;"), each measurement is reduced for one hour during wintertime and two hours during the summertime period. The variable *pollInterval* can be used if the respective device should to be polled in a periodic manner (e.g. "pollInterval 3600;"). Desired intervals must be defined in seconds.

To add a new connector, the following methods must be implemented:

- *getSourceData()* – Returns the latest measurement.
- *getSourceData(starttime)* – Returns all measurements after the requested *starttime*. If historical reads are not supported, only the latest measurement should be returned.

If the desired connector supports write access (actor), the method *writeData(value)* must be implemented too. The actual measurement processing is implemented within the methods *processEvent()* and *processTimeframe(starttime)*. If the respective connector reads its measurements event triggered, the method *processEvent()* or *processEvent(measurement)* must be called for each event. If polling is enabled (by defining the variable *pollInterval*), the method *processTimeframe(starttime)* is called by the *PollService* within each interval.

Storing measurements

The overall communication from the connectors to the MOST toolkit is transacted within the class *ConnectorService.* Currently, all measurements are stored directly in the MOST database using the Java *DataSource* interface. This forces the toolkit to periodically poll measurements from the database for processing applications requesting real-time data access. To avoid this communication overhead and improve real-time event propagation, the class *ConnectorService* is intended to be updated. By using web services to store measurements, events can be triggered within the data-abstraction framework for each new value (see chapter 3.3). This

enables the toolkit to directly notify processing applications about new measurements, without polling the database.

3.1.2. Java Database Connectivity - Connector

This connector is based on the proposed framework and enables communication with systems and read/write file formats, which are supported by Java Database Connectivity (JDBC) compatible drivers. It therefore supports data access to various databases (Oracle, Microsoft SQL, ODBC compliant databases, etc.) and popular file formats such as CSV, Excel, etc. based on the Structured Query Language (SQL). Due to technical limitations (the JDBC library is not notified when new data is added to the database/file), this connector supports data transfer by polling in a periodical manner only (i.e., every minute/hour/day/week/etc.). A class diagram of the actual implementation is shown in Figure 4.

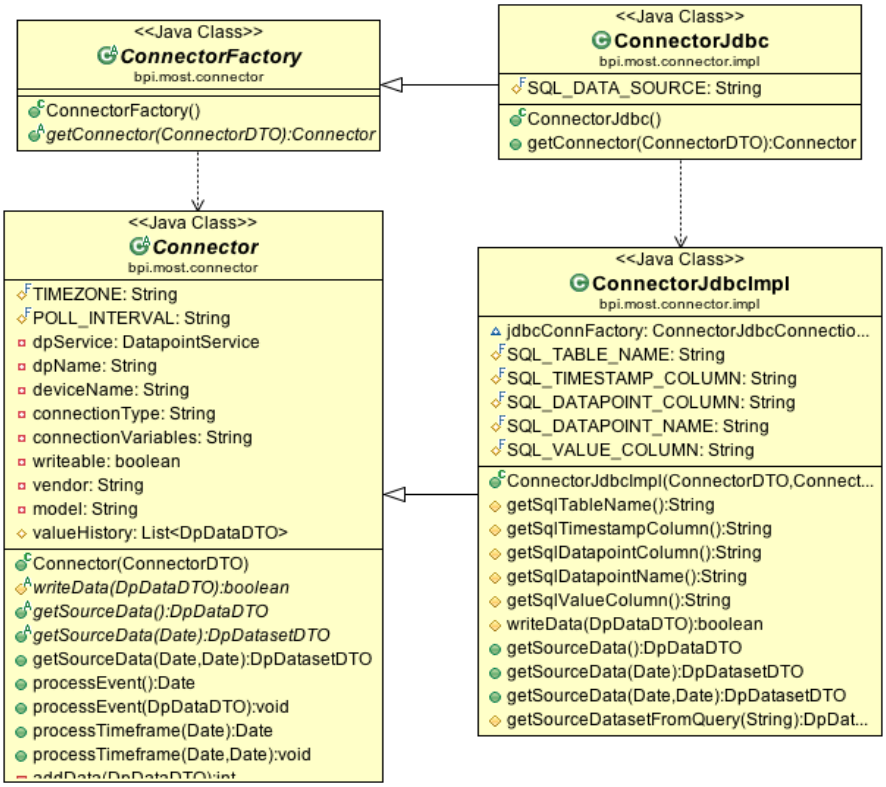


Figure 4. Class diagram of the JDBC connector implementation

The proposed JDBC connector is used for all data sources that define the connection type “jdbc” and provide no vendor and model information. The class *ConnectorJdbc* implements the *ConnectorFactory* and checks the requested connection type, vendor and model. The connector variable *sqlDataSource* is used to find the respective JDBC compatible source (database or file). If the desired data source is available, the actual connector implementation *ConnectorJdbcImpl* is instantiated.

The class *ConnectorJdbcImpl* extends the abstract class *Connector*. It automatically detects the table structure of the data source based on the information described in Table 5. By probing the defined column names, appropriate SQL statements are generated. This enables plug and play support for most common data source structures shown in Table 6 (a) and (b).

Table 5. Variables required for the JDBC connector

Datasource definition	Description
sqlDataSource	Name of the DataSource connection (database connection) registered to the respective Java Virtual Machine.
sqlTableName	Name of the table including the measurements
sqlTimestampColumn	Columnname of timestamp
sqlDatapointColumn	Columnname of the datapoint/sensor
sqlDatapointName	ID of the datapoint (required only if all datapoints are in the same column)
sqlValueColumn	Name of the column containing the value (required only if all datapoints are in the same column)

Table 6. Supported data source structure of the JDBC Connector

a) Multiple columns

Timestamp	Sensor 1	Sensor 2	etc.
2012-01-01-15:00	23,5	14,1	…
2012-01-01-15:10	22,3	14,4	…

b) Multiple rows

Timestamp	Sensor (ID)	Value
2012-01-01-15:00	Sensor 1	23,5
2012-01-01-15:00	Sensor 2	14,1
2012-01-01-15:10	Sensor 1	22,3
2012-01-01-15:10	Sensor 2	14,4

3.1.3. OPC Data Access - Connector

The Open Process Control (OPC) Data Access (DA) standard (OPC 2012) enables communication to various building automation networks such as BACnet, KNX, EnOcean, M-Bus, ZigBee, as well as to many building management systems (BMS). Due to implementation complexity, the OPC connector is implemented with the programming language Gamma, using the OPC Datahub (OPC Datahub 2012) environment. All systems providing a OPC DA Server software can be accessed by this connector (see Figure 5). Since all OPC DA Servers provide the same Application programmable Interface (API), uniform communication to different building networks is possible. OPC DA supports registration for events (value changed, etc.). This allows to process building data streams in real-time.

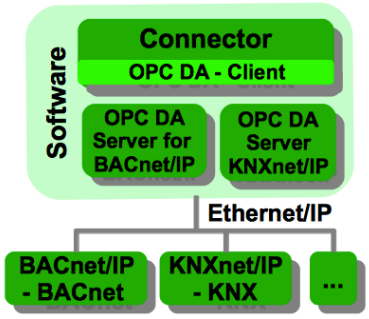


Figure 5. OPC Client / Server infrastructure

The class diagram of the current implementation is shown in Figure 6. Each project inherits from the *ConnectorEngine*. On startup, a *DatapointOPC* object is instantiated for each datapoint with an OPC data source defined in the MOST database. Adaption of measurements within the Connector (e.g. to convert the measurement into a desired format) can be achieved by overwriting the *transformValue()* method.

Because of the limited adaption options and vendor lock risks caused by using the proprietary environment OPC Datahub / Gamma scripting, a technology independent alternative is envisioned. Within the Google Summer of Code 2012 (GSOC 2012), the development of a Java based OPC Connector, using the OPC DA library JEasyOPC 2012 and the proposed connector framework, is intended.

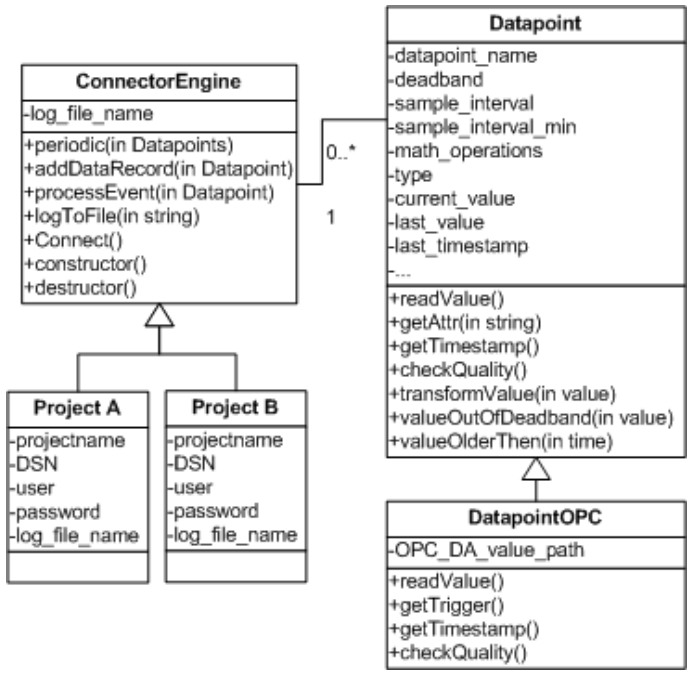


Figure 6. Class diagram of the OPC DA connector

3.1.4. Remote data collection

Since all communication of the presented connectors to existing building systems is based on Ethernet/IP, various installation setups are possible. For example, secure remote access to different building systems can be realized by using a Virtual Private Network (VPN). Figure 7 shows a configuration setup where the monitoring server collects data from various building systems (located anywhere in the world) through a VPN.

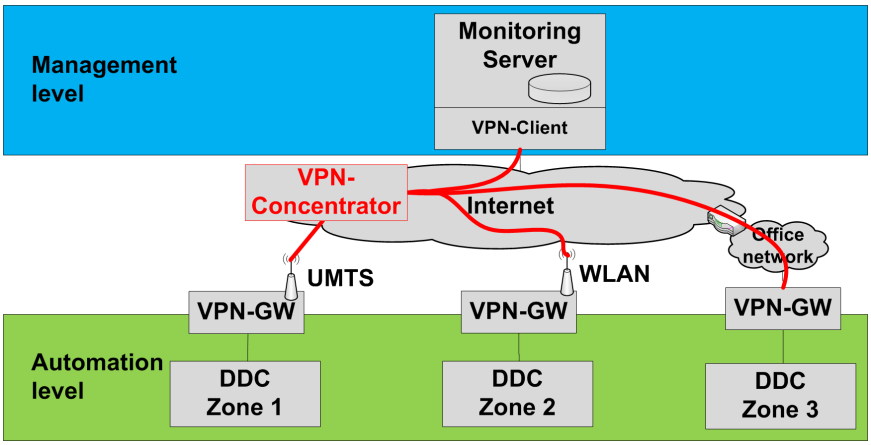


Figure 7. Using a Virtual Private Network for monitoring over building limits

By using an adaptive routing configuration on the VPN-GWs (e.g. OpenWrt 2012), infrastructure independent plug and play installation is possible. For example, the VPN-GWs can scan for possible Internet connections (Ethernet/IP, WLAN, UMTS, etc.) and automatically connect to the best fitting one. Based on the VPN a secure communication to the monitoring server is guaranteed. By using watchdog scripts on the VPN-GWs, automatic reconnection in case of communication faults can reduce data loss and maintenance effort.

3.2. Database

The MOST database is designed to historically store and process various different building data streams. Storage rules minimize database load while a number of data preprocessing functions enable effective data analysis. Various performance optimizations minimize required computer resources. Performance benchmarks are conducted to show the practical limits.

Since large building complexes can include many thousand measurement points (temperature, occupancy, energy use, etc.), a scalable database technology is required for historical data storage. The data transfer rate of most building data sources is comparable to other domains (web applications, ticket systems, etc.). Data requests can cover a few hours to multiple years and are often limited to particular datapoints or zones under specific conditions. Appropriate database technologies can be categorized in relational databases (Kemper and Eickler 2009) and No Structured Query Language (NoSQL) solutions. Relational databases structure data within an Entity Relationship (ER) model and provide data access with the Structure Query Language (SQL). NoSQL technologies use no uniform storage structure, provide very diverse data access interfaces, but can sometimes be optimized to accommodate specific use cases. Due to available wide support in different software environments, the adoptability, and the documented performance characteristics, the open-source relational database MySQL was chosen.

3.2.1. Database-design

The developed ER model shown in Figure 8 focuses on a generic representation to support various building data sources. All measurement points are expressed in the *datapoint* table and hold information about

- the location of the measurement (by referencing a zone)
- if it is a virtual or physical sensor/actor (by referencing a data source)
- the unit, accuracy, value range, deadband, sample interval, etc.

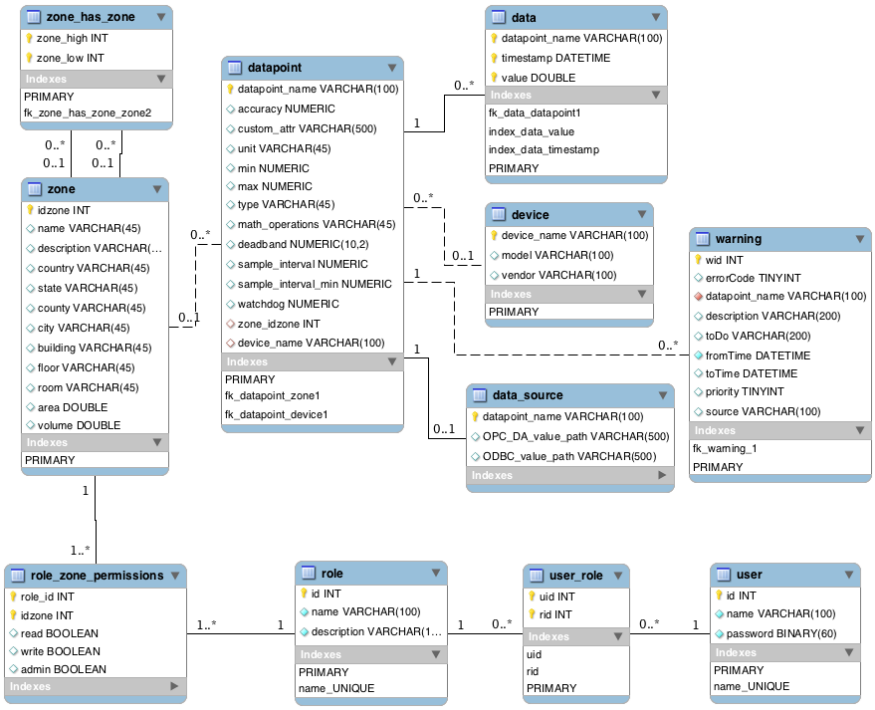


Figure 8. Entity-relationship model of the proposed database

Multiple *datapoints* can be aggregated into a physical *device*. Measurements are stored within the *data* table by adding a new row with *datapoint name*, *timestamp,* and *value*. The Coordinated Universal Time (UTC) is used as *timestamp* to prevent overlapping data sets during the switch of summertime and wintertime. The *datapoint_name* field is the

identifier of the sensor/actor and is unique for each datapoint. The *value* field contains the measured data represented in units defined in the *datapoint* table. Any values compatible to the double format are supported. Boolean values are expressed as 1 (true) and 0 (false). The source of each *datapoint* is defined in the *data_source* table. Each *datapoint* can generate a message in the *warning* table in case of a malfunction. The *zone_has_zone* table enables *zones* to be grouped independent of any hierarchy. The storage engine InnoDB is used in the default configuration. InnoDB provides row-locking, which enables multiple queries to insert data without locking the whole table. These feature increase multi-user concurrency and performance. Using the proposed database design, sensors/actuators can be easily added and removed during the building's lifecycle.

3.2.2. Data preprocessing

Various performance optimization techniques are used to minimize the required computer resources and to improve database performance. This includes Indexes to cache measurements in the memory and MySQL Stored Procedures to implement various data processing algorithms.

Indexes

To improve the performance of reading stored measurements, the *data* table is highly indexed. It uses a multiple-column hash-index based on the *datapoint name, timestamp,* and *value* to presort measurements. The index is kept in memory and enables the database to process datapoint and timeframe specific requests (get values of datapoint X from time A to time B, etc.) without accessing the hard disk. By including the *value* column in the index, the system can return stored measurements without accessing the hard disk at all (MySQL 2012). To reduce memory usage, varying index approaches can be applied to different MySQL partitions as explained further down. By using an appropriate index design, a significant data access performance improvement can be reached.

Stored procedures

All access to the database is done with „getters“ and „setters“ implemented in MySQL stored procedures. This object-oriented way of database access allows handling performance and permission issues in a more fine-grained way than with direct access using SQL. It also enables a centralized implementation of data preprocessing algorithms. This can optimize data-query performance and prevent redundant code in different client applications. Table 7 and Table 8 list some of the developed stored procedures.

Storing measurements

To store measurements, the stored procedure *addData(dpName, timestamp, value)* is provided. Since measurements of diverse building data streams can occur event based or periodically, a generic data storage strategy was developed (see Figure 9). When a value of a measured parameter is added, the system checks if the value is out of a defined *deadband* and a minimal sample interval (*sample_interval_min*). The measurement is only stored if both criteria are fulfilled. Thus, flooding of the database with unnecessary data is avoided. If a *sample_interval* is defined for a specific datapoint, a value is saved at each period. All time frame definitions are done in seconds. The *deadband* is defined in the unit of the respective datapoint. To ignore the proposed processing rules the *addDataForced(dpName, timestamp, value)* can be used. This data storage strategy maximizes temporal accuracy while minimizing the database load. Due to the minimized database size, measurements can be processed in an efficient manner.

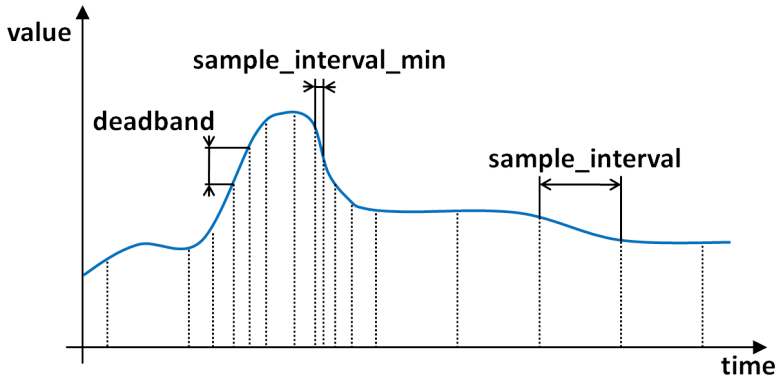


Figure 9. Data storage rules to minimize database load

Reading measurements

To simplify data analysis, powerful data preprocessing algorithm for data requests were developed. For example, window state information can be stored in a high-resolution fashion by saving timestamps marking the window opening and closing actions. Subsequent data processing applications may want to obtain this information in a periodic manner (e.g., hourly). Therefore, the preprocessing algorithm must deliver, for each discrete interval, either the value "open", or "close". This is achieved via appropriate reasoning depending on the use case of the processing application. For example, window may be declared open or close if a corresponding action took place in the respective interval. Alternatively, window may be declared open if it was open during most of the respective interval. To account for this and other data preprocessing challenges, a number of stored procedures (e.g. *getValuesPeriodicXXX*) are implemented. Figure 10 to Figure 12 show examples of values calculated with the stored procedure *getValuesPeriodicBinary(dp, start, end, period, mode)* using different modes. Crosses mark stored measurements while circles show calculated return values. Figure 13 and Figure 14 show the modes provided by the stored procedure *getValuesPeriodicAnalog(dp, start, end, period, mode).*

Watchdog

To probe the database for datapoints, which do not deliver measurements in the expected timeframe, a watchdog service was developed. Based on the information defined in the column *watchdog* of the *datapoint* table, the stored procedure *runWatchdog()* examines missing measurements. If a datapoint does not store any measurements for the defined *watchdog* interval, a corresponding message in the *warning* table is generated. The *runWatchdog()* function is called every 15 minutes in the default configuration.

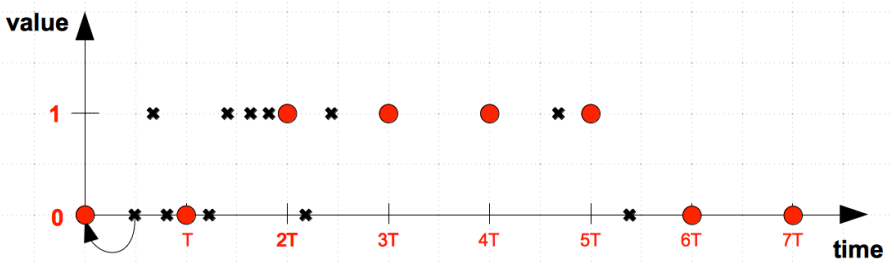


Figure 10. getValuesPeriodicBinary() - mode 1: majority decision / sample & hold

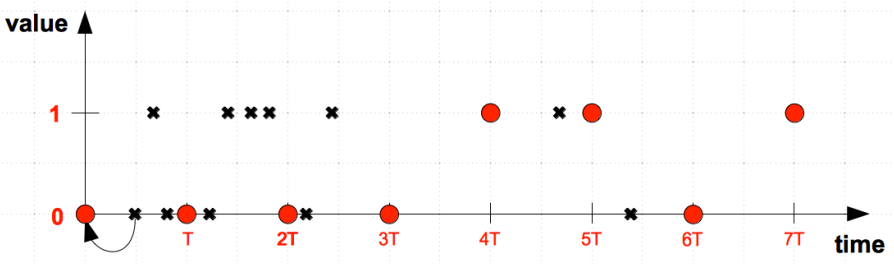


Figure 11. getValuesPeriodicBinary() – mode 2: forced 0 / default 1

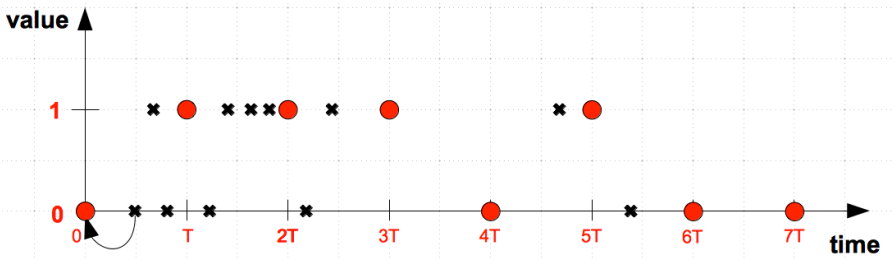


Figure 12. getValuesPeriodicBinary() – mode 2: forced 1 / default 0

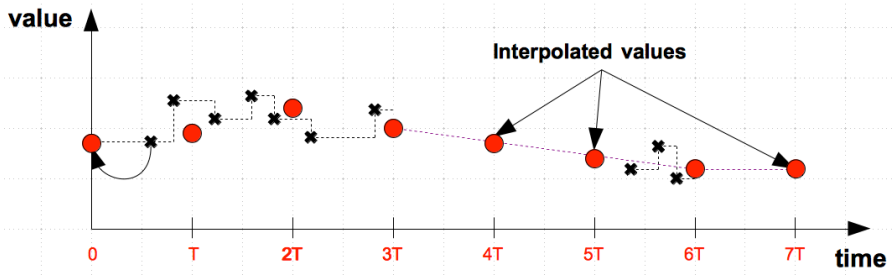


Figure 13. getValuesPeriodicAnalog() – mode 1: time-weighted average / linear interpolation

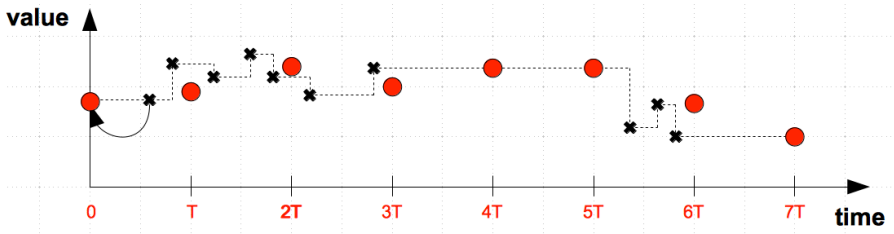


Figure 14. getValuesPeriodicAnalog() – mode 2: time-weighted average / sample & hold

Table 7. Data-preprocessing with stored procedures

Name	Description
getValues(*dp*, *start*, *end*)	Returns stored measurements of the requested datapoint (*dp*) between the *start*- and *endtime*. If the argument *starttime* and *endtime* are null the last entry is returned. If only the argument *endtime* is null the first measurement before *starttime* is returned. If only the argument *starttime* is null the first measurement after the *endtime* is returned.
getValuesWhereDpEqual(*dp1*, *start*, *end*, *dp2*, *value*)	Returns all measurements of datapoint 1 (*dp1*) where the nearest value in the past of datapoint 2 (*dp2*) is equal to *value*. Additional functions with lower, bigger and between conditions are available.
getValuesPeriodic*(dp, start, end, period)*	This procedure automatically selects and calls the required function (analog or digital) for the generation of values, depending on the type of the datapoint (*dp*). *Period* specifies the time interval of the returned values. A quality index gives feedback about how many real measurements are available for the calculated values at each interval.
getValuesPeriodicAnalog(*dp, start, end, period, mode*)	Returns periodic values for any type of analog measurement (temperature, carbon dioxide, relative humidity, etc.). In the default *mode (1)*, a linear interpolation and arithmetic average is used for calculating periodic values. If the requested period contains more then one measurement, the arithmetic average is calculated. If no measurement is available for the requested period, a linear interpolation to the next measurement is performed. Mode 2 uses sample & hold instead of linear interpolation.
getValuesPeriodicBinary(*dp, start, end, period, mode*)	Returns periodic values for any type of digital measurement (window/door state, occupancy, etc.). It supports three *modes*: - majority / sample & hold, - dominant "0" / default "1", - dominant "1" / default "0". The majority/sample & hold mode returns the majority if more then one measurement is available in the requested period. If no measurement is available, the last value of the previous period is returned. The mode dominating "0"/default "1" returns "0", if one or more measurements in the requested period are "0". If no measurement is available, the default value "1" is returned. The mode dominating "1"/default "0" works the same way, but swaps "0" and "1".
getValuesPeriodicWhereDpEqual(*dp1*, *start*, *end*, *period*, *dp2*, *value*, *modeDp1*, *modeDp2*)	Returns all measurements of datapoint 1 (*dp1*) where the datapoint 2 (*dp2*) has the equal *value*. *ModeDp1* and *modeDp2* is used for generating the respective periodic values. Additional functions with lower, bigger and between conditions are available.

Table 8. Datapoint, device, and zone management

Name	Description
runWatchdog()	Probes all datapoints if measurements are delivered in the expected timeframe. A warning is created in the warning table if a fault is detected.
addDatapoint(*dpName, type, unit, accuracy, min, max, deadband, sample_interval, sample_interval_min, watchdog, math_operations, custom_attr*)	Adds a datapoint with the respective characteristics to the datapoint table. The *dpName* is needs to be unique within the database. Type is a freely choosable string. If appropriate one of the following should be selected: temperature, humidity, air-quality (CO2, VOC), state (contact sensors, etc.), meter (heat meter, electricity meter, etc.), power, occupancy, brightness, other. The arguments *math_operation* and *custom_attr* are not implemented yet and can be set null.
addDevice(deviceName, vendor, model)	Adds a device with its respective *vendor* and *model* description, *DeviceName* needs to be unique within the database.
addZone(*name, description, coutry, state, county, city, buiding, floor, room, area, volume*);	Adds a zone with its respective description. Not used parameters can be set null.
addDatapointToDevice(*dp*, *deviceName*);	Adds the datapoint *dp* to a device (*deviceName*).
addDatapointToZone(*dp*, *zone*);	Adds the datapoint *dp* to a *zone*. The ID of the *zone* needs to be passed.
addZoneToZone(*zone1*, *zone2*);	Adds an existing zone (*zone1*) to another existing zone (*zone2*). The IDs of the zones needs to be passed.
deleteDatapoint(*dp*)	Deletes the datapoint dp from the table datapoint, all associated measurements and the definition of the data source.
emptyDatapoint(*dp*)	Deletes all measurements of the datapoint *dp*, but keeps the datapoint and data source table untouched.
emtpyDatapointTimeslot(dp,start,end)	Deletes all measurements of the datapoint *dp* between *start*- and *endtime* and but keeps the datapoint and data source table untouched.

3.2.3. Setup options

The proposed database design supports various installation setups. The following section describes a few possibilities to improve the database performance.

MySQL configuration parameters

Since the size of the data table can reach several GBytes in case of large implementations, the MySQL index pool size should be increased to minimize hard disk operations. For the default storage engine *InnoDB* the *innodb_buffer_pool_size* parameter in the *my.cnf* file should be set depending on the database size. Given a dedicated database machine, up to 80% of the available memory can be used for the index pool. For the storage engine *MyISAM* the *key_buffer_size* parameter must be used. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

MySQL partitions

Using partitions enables splitting of measurements in different *virtual tables* as shown in Figure 15. Values are separated depending on their timestamps. From the user point of view, only the *data* table is visible. This strategy improves database performance by using diverse index approaches on different partitions. Recent partitions can be highly indexed while older partitions use fewer indexes to reduce memory usage. The overall reduced memory usage improves performance for highly loaded databases.

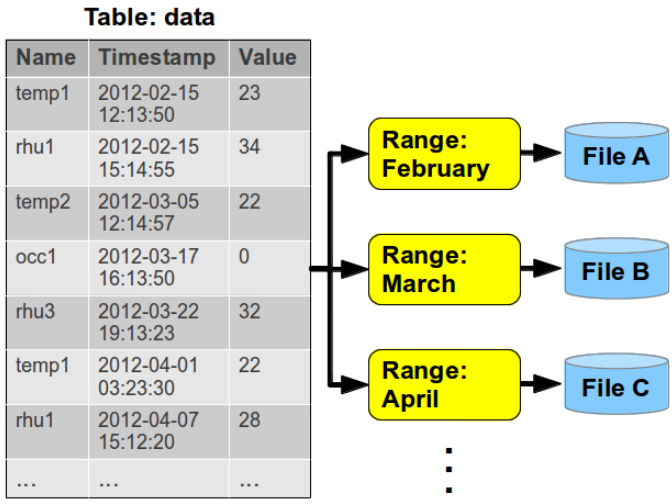


Figure 15. Splitting measurements in different partitions depending on their timestamps

MySQL replication

The default database configuration performs well for most common queries. Specific use cases (large batch processing, long-term data archiving, etc.) can be optimized by using MySQL replication with adapted partition and index layout as shown in Figure 16. This setup is only required for large installations with diverse usage patterns.

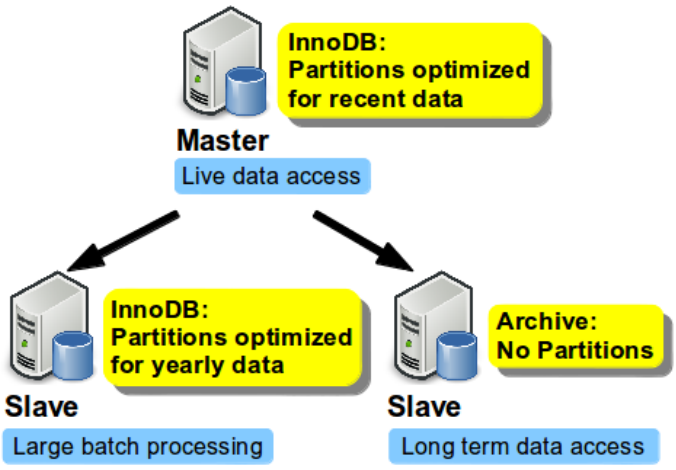


Figure 16. Using MySQL replication to optimize the database for different use cases

3.2.4. Benchmark tests

Using a simulator implemented with the programming language Java, different benchmark tests were conducted. The tests were executed in a virtual machine using Virtualbox with four Intel Xeon CPU X5650 cores at 2.67GHz and 6 GByte Memory. The host and the guest system were running a 64 bit Debian 6.0 Linux Distribution and MySQL version 5.1.49 was installed for testing.

The following test cases were selected to compare the performance of the database design with varying number of datapoints, zones, and stored measurements. All tests are conducted for two different scenarios. Scenario A consists of 10000 datapoints, 2500 zones, and 2.5 million stored measurements. Scenario B consists of 100000 datapoints, 25000 zones, and several million stored measurements.

Testcase 1.x

These test cases tread the write performance of the database system. Therefore, five concurrent connections are established to the database, and each connection writes 10000 measurements into the data table as fast as possible.

Testcase 2.x

These test cases tread the read performance of the database system. They cover:

- reading of raw data - getValues()
- generation of periodic values - getValuesPeriodic()
- reading raw data with conditions - getValuesWhereDpEqual(), etc.
- generation of periodic values with conditions – *getValuesPeriodicWhereDpEqual()*, etc.

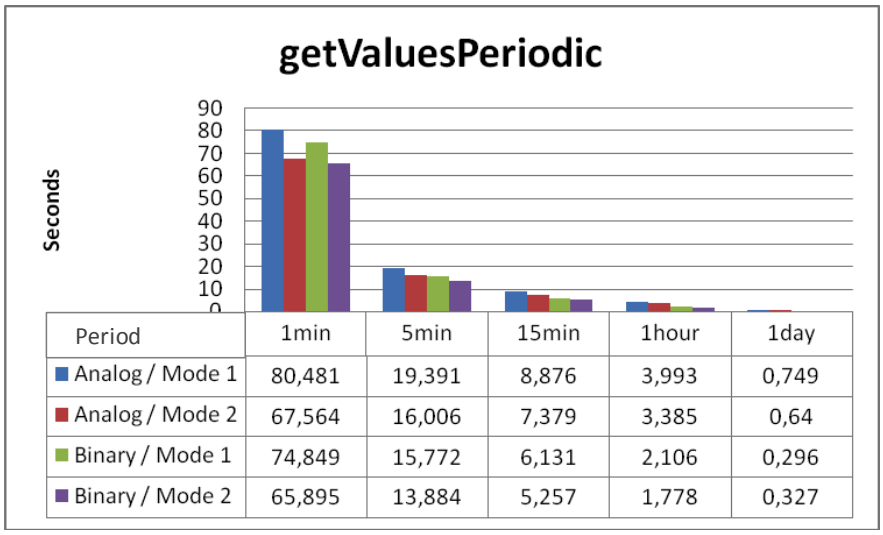
Testcase 3.x

These test cases tread the read performance under concurrent write load of the database system. Therefore the same test cases as in 2.x are run, while a new measurement is written every 50ms.

Results

The number of defined datapoints showed no significant impact on the performance of different stored procedures. The overall performance of the developed preprocessing algorithm demonstrates the usability of the presented storage approach. For example, the calculation of hourly values of a common room temperature sensor within a timeframe of 6 month takes about 3 seconds in the described benchmark setup.

Test cases 1.x showed a maximal writing performance of about 1000 new measurements per second. Test cases 2.x compared the performance of the stored procedures using different conditions (timeframes, modes, periods, etc.). Figure 17 shows the duration of generating periodic values depending on the period, the type of the datapoint, and the mode used.

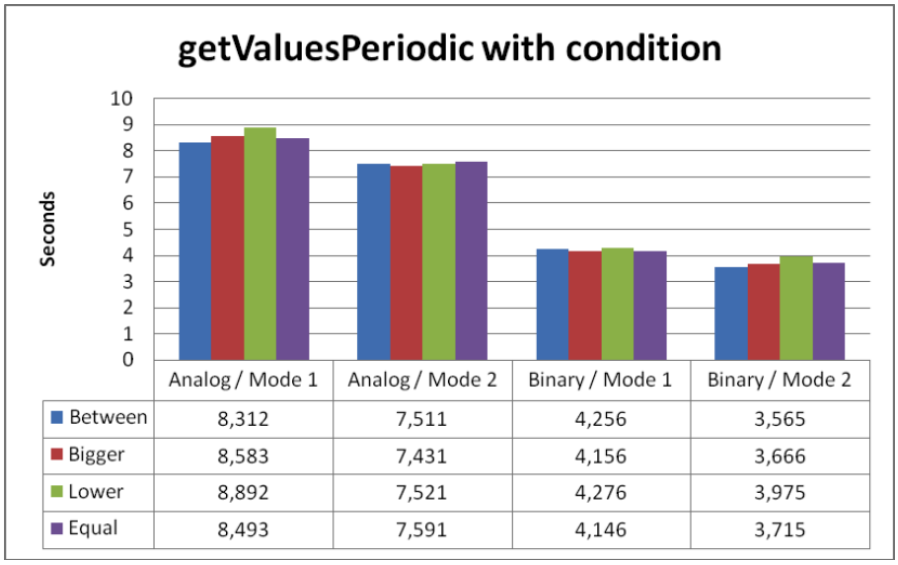


Period	1min	5min	15min	1hour	1day
Analog / Mode 1	80,481	19,391	8,876	3,993	0,749
Analog / Mode 2	67,564	16,006	7,379	3,385	0,64
Binary / Mode 1	74,849	15,772	6,131	2,106	0,296
Binary / Mode 2	65,895	13,884	5,257	1,778	0,327

Figure 17. Duration of calculating periodic values depending on period, datapoint type and mode

The timeframe of the queries is 6 month. The processing time of different queries (Analog/Binary, Mode 1 / 2 and varying periods) is shown in seconds and consists of the calculation time only. The time needed to fetch the calculated values from the database to the processing client is not included, since it varies with different installation setups.

The calculation of periodic data turns out to be much faster for the longer periods as compared to the shorter periods. This is because averaging is much faster than interpolation, so if there is at least one value in each period, the whole stored procedure call is much faster. Furthermore, the used mode has little impact on the calculation duration, although in general binary datapoints are calculated faster than analog ones and mode 2 is faster than mode 1. Figure 18 shows the duration of generating periodic values, depending on the value of another datapoint (e.g., *getValuesPeriodicWhereDpEqual*).



	Analog / Mode 1	Analog / Mode 2	Binary / Mode 1	Binary / Mode 2
Between	8,312	7,511	4,256	3,565
Bigger	8,583	7,431	4,156	3,666
Lower	8,892	7,521	4,276	3,975
Equal	8,493	7,591	4,146	3,715

Figure 18. Duration of calculating periodic values depending on condition type, datapoint type and mode (period = 1hour)

The generation of the periodic values takes most of the stored procedures processing time. Therefore, the duration of a stored procedure call is

much smaller for binary datapoints then for analogue ones. Moreover, the duration does not depend on the condition type used (between, higher, lower, equal). Test cases 3.x showed no significant impact of concurrent read and write access on the database.

3.3. Data-Abstraction Framework

The Data-Abstraction Framework enables consistent access to different building data streams. It supports various data aggregation functions (e.g. virtual datapoints), covers user authentication and zone management, and provides different software interfaces for real-time data processing with diverse client applications (Web interface, Microsoft Excel, MATLAB, etc.). Figure 19 shows the communication between the abstraction framework (server side) and different processing applications (client side). The programming language Java is used for implementation of the proposed abstraction layer.

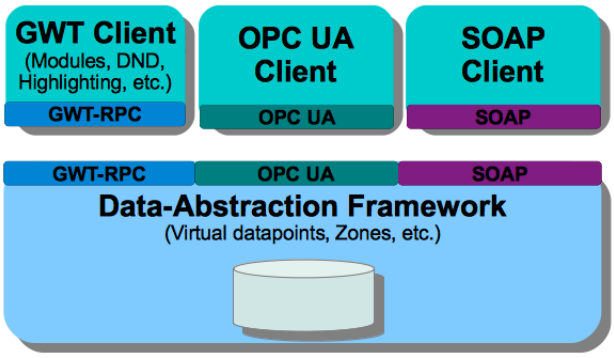


Figure 19. Communication between server and client side

3.3.1. Physical and virtual datapoints

The server side abstraction layer represents all sensors and actors as datapoints. The abstract class *Datapoint* is used to provide a coherent view to desired datapoints. Figure 20 shows a class diagram of how datapoints can be accessed within the framework.

Two distinct implementations of the class *Datapoint* are available in the framework (*DpPhysical* and *DpVirtualXXX*). The class *DpPhysical* represents real sensors and actors of a building and reads/writes directly to the database. So-called "virtual datapoints" are used to obtain

information concerning parameters that are not directly measureable. A virtual datapoint can be, for example, the energy use of a zone, which represents an aggregation of all energy measurements in this zone. Virtual datapoints can be added by implementing the abstract class *DpVirtualFactory* and register it to the Java Service Loader (by using the text file META-INF/services/bpi.most.server.DpVirtualFactory). Each implementation of a virtual datapoint has to provide a unique identifier (String ID) and need to return a desired *Datapoint* implementation with the method *getVirtualDp(String uniqueString)*. On request, the *DpController* searches for the respective implementation and returns a reference to an instance. Generic virtual datapoints are provided within the framework. A prototypical implementation of a virtual datapoint is shown in appendix 7.2.

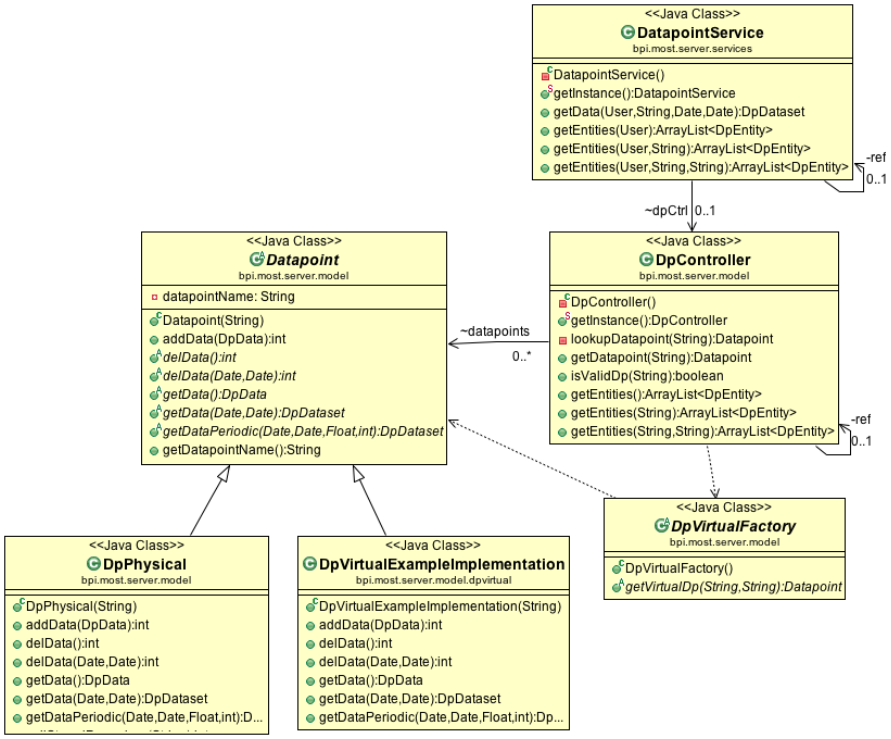


Figure 20. Building data abstraction using different Datapoint implementations

From the user's point of view, the same mode of access (abstract class *Datapoint*) can be applied to both physical (sensor-based) and virtual datapoints. The data aggregation functionality provided by virtual datapoints enables the reduction of the vast amount of measured data in buildings, to the respective information required. By using the data preprocessing functionality of the proposed database, information processing is accelerated. For example, the calculation of temporally structured data sets, e.g. hourly/weekly/monthly/etc. values, linked queries of energy use for specific time intervals and building zones under specific occupancy conditions, etc. are implemented within the database. This approach increases the performance and scalability of the overall system.

3.3.2. Measurements

Each measurement is represented as a *DpData* object within the framework. It includes the measured value, the timestamp and a quality value to judge the correctness of the measurement. A set of measurements within a specific timeframe is composed to a *DpDataset*. The class *DpDataset* is implemented as an *ArrayList* of *DpData* objects as shown in Figure 21. In addition, several methods are provided by the *DpDataset* class to simplify data processing, e.g. *getDataBefore(timestamp)*, *getDataAfter(timestamp)*, etc.

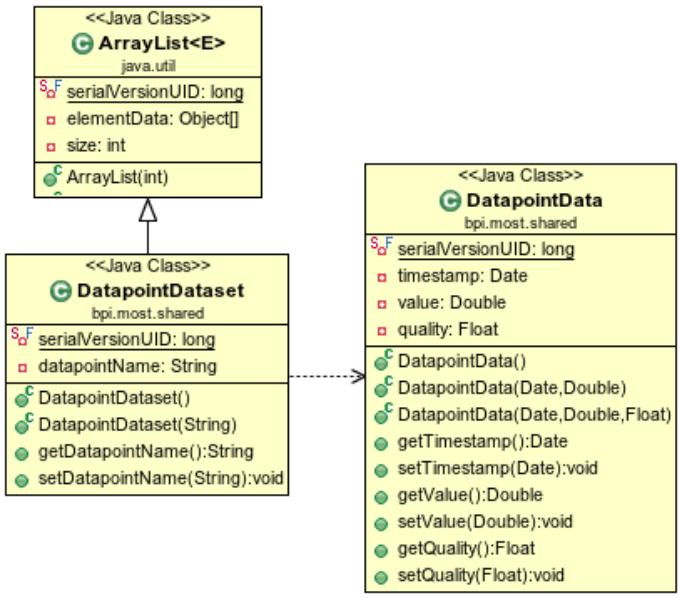


Figure 21. Measurement representation within the framework

3.3.3. Zone management

To group multiple datapoints within zones, the classes *Zone*, *ZoneController* and *ZoneService* are developed (see Figure 22). These classes enable logical grouping of datapoints independent of any physical structures (stories, rooms, etc.). The current implementation limits a datapoint to be linked to one zone only. However, a zone can be part of multiple other zones and a zone can contain any number of subzones and datapoints.

The method *getHeadZones()* returns the head objects of defined zone tree structures. Subsequent, the *getSubzones(int sublevels)* method can be used to browse the tree. The variable *sublevel* defines the number of requested sublevels. The leaves of the zone structure are datapoints and can be requested with the method *getDatapoints(int sublevels)*.

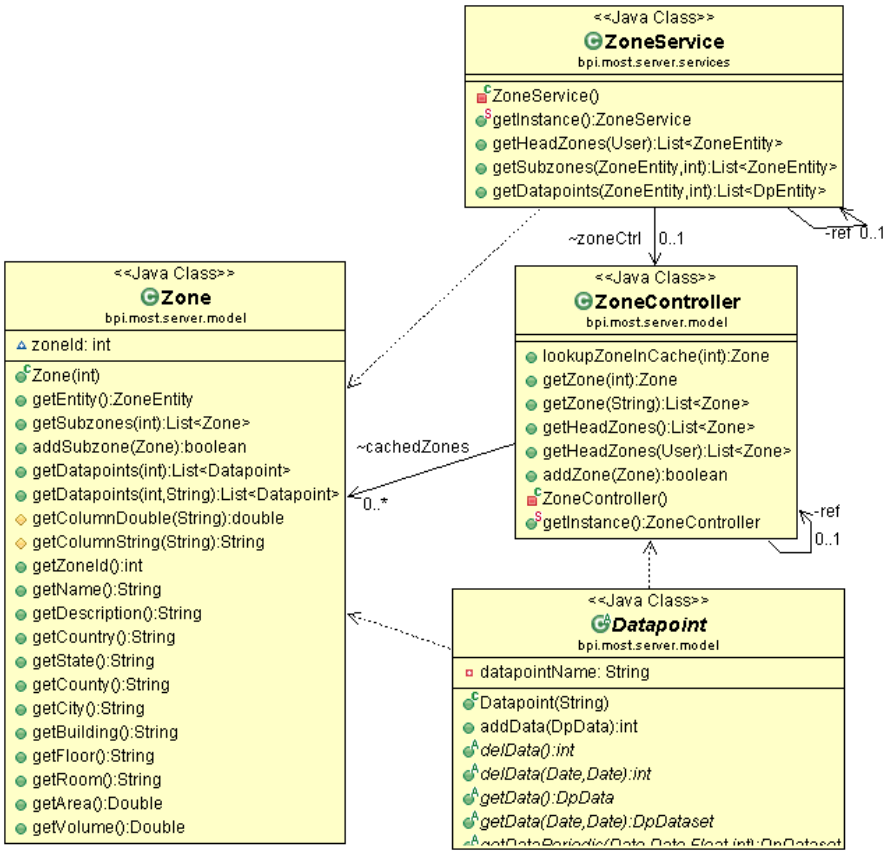


Figure 22. Zone management within the data-abstraction framework

The class *Zone* is used as Data Access Object (DAO) to zone information. This enables future development to integrate other zone definition sources into the abstraction layer. For example, Building Information Models (BIM) could be used to provide zone information by implementing a respective DAO zone class. In case of IFC2x4 2010, this DAO class could use the IFC objects *IfcZone*, *IfcSpatialZone*, *IfcSpace*, etc. to gather required information.

3.3.4. Database connection pool

To support scalable access to the database, the tomcat JDBC connection pool (Tomcat 2012) is used. The class *DbPool* implements the interface to the pool using a singleton design pattern. Figure 23 shows the configuration options and the methods provided by the connection pool.

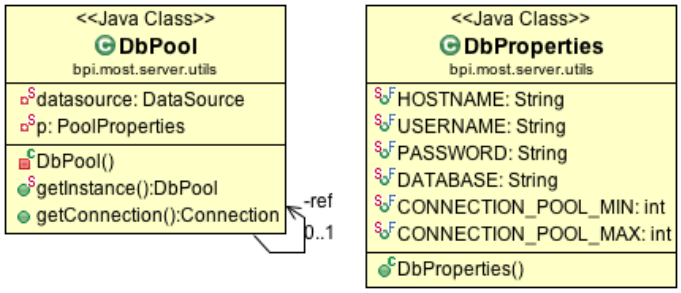


Figure 23. Database connection pool implemented in the class DbPool

3.3.5. Remote data access

The proposed abstraction layer provides different software interfaces for data access. To support communication with the Google Web Toolkit (GWT) the GWT Remote Procedure Call (RPC) protocol is used. To enable direct access from additional processing applications (e.g. Microsoft Excel), an OPC Unified Architecture (UA) server and a web service, based on the Simple Object Access Protocol (SOAP), is currently in development. In addition, a data export for diverse file formats is planed to be implemented within the web visualization framework, described in chapter 3.5.

All software interfaces (GWT-RPC, SOAP, OPC UA) provided by the abstraction layer are part of the package *bpi.most.server.services*. This package contains generic service classes and an additional sub package for each interface implementation as show in Figure 24. The generic service classes (*AuthenticationService*, *DatapointService*, *ZoneService*) provide access to the abstraction model (*Datapoint*, *DatapointController*, *Zone*, etc.) and cover permission issues. Each interface implementation must provide an authentication service and map respective sessions to *User* objects. Subsequent, this *User* objects are passed to the service methods for each request (see Figure 25).

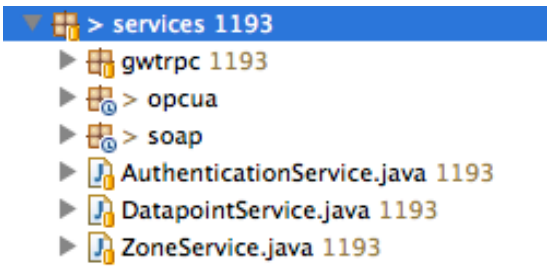


Figure 24. Generic service classes

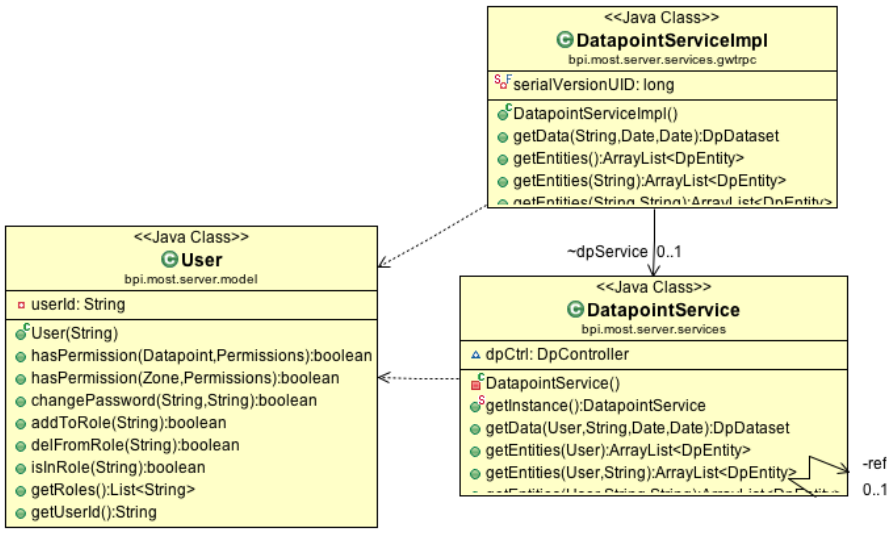


Figure 25. Generic datapoint service used by the GWT-RPC implementation

Javadoc 2012 is used to document the developed source code. Figure 26 gives an overview of available packages and classes implemented in the data-abstraction framework. A current version of the documentation is available at http://most.bpi.tuwien.ac.at/doc/javadoc-server/index.html

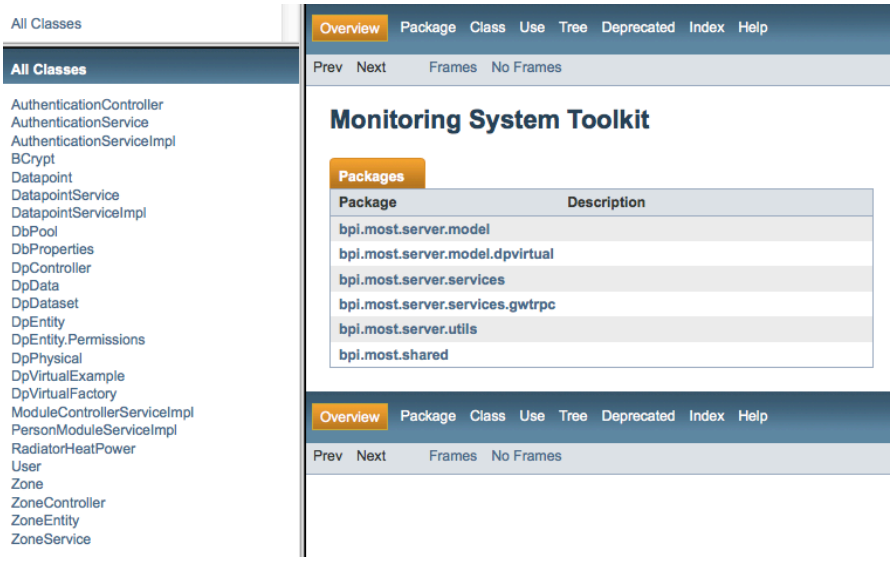


Figure 26. Source code documentation overview

3.4. Data-Processing Framework for MATLAB

To simplify complex data processing with MATLAB, a dedicated framework was developed. It provides common data processing algorithms in an object-oriented way. Figure 28 shows a cutout of the class diagram. The complete class diagram is shown in the appendix 7.2. The *MainCtrl* initializes the framework (instantiates required objects – Datapoint's, StatisticCtrl, etc.) and keeps track of the database connection. A set of Objects (*StatisticCtrl*, *PlotCtrl*, *DatapointCtrl*, *Datapoint*, *ZoneCtrl*, etc.) provide various methods to simplify data analysis. Figure 27 shows a sample psychrometric chart generated with the method *plotMollierOfZone()*. Due to the open-source nature of the framework, functionality can be easily enhanced or adopted to respective needs.

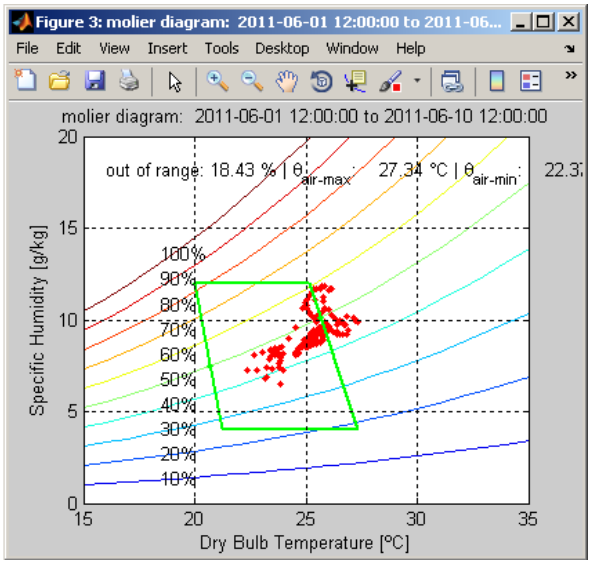


Figure 27. Psychrometric chart generated with the MATLAB-Framework

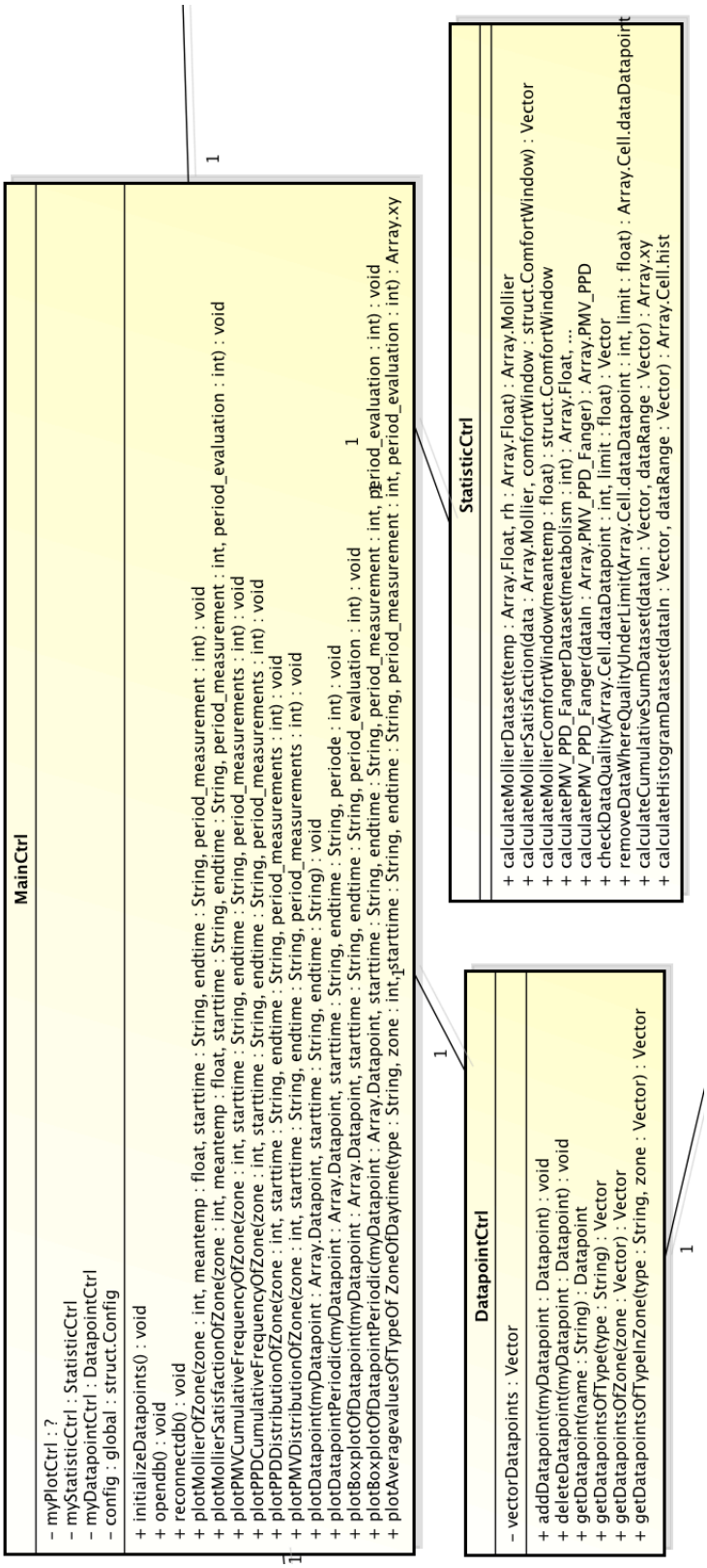


Figure 28. Cutout of the MATLAB-Framework class diagram

3.5. Visualization Framework

The proposed framework simplifies platform independent graphical user interface (GUI) development for building data visualization. Various components are provided to reduce development efforts. Exemplary modules show the usability of the framework.

3.5.1. Technical requirements

Different native GUI toolkits support different platforms (Linux, Windows, OSX) and have different features such as accessibility, layout engines, and looks. To enable platform independent GUI development, web based technologies can be used. Furthermore, web frameworks (JSF, GWT, etc.) simplify development by abstracting specific web technologies. An overview of common standards, programming languages and libraries is shown in Table 9 to Table 11.

Due to the platform and technology independence of web frameworks, the Google Web Toolkit (GWT) was chosen for implementing the proposed visualization framework. GWT enables web development by using the programming language Java for server (running on a central server station) and client side code (running in the user's browser). Client side code is converted to platform optimized JavaScript at compile time. This strategy allows the development of reusable components for several use cases.

Table 9. Visualization technologies based on native software solutions

Common PC solutions	Mobile devices (smartphone, tablet, etc.) solutions
Linux: - GTK (used in Gnome) - QT (used in KDE) - EFL (used in Enlightenment)	Android: - Java based SDK
Mac OSX: - Cocoa	iOS: - Objective-C based SDK
Windows: - WPF - Metro (Windows 8)	Windows Phone 7: - Metro
OS independent: - Java based libraries (e.g. Swing)	OS independent: - Java ME

Table 10. Common web standards

Communication standards	Visualization standards
- HTTP 1.1 - HTTP 2.0, SPDY - Ajax - Web services (SOAP)	- HTML, CSS - WebGL

Table 11. Web based visualization technologies

Server side	Client side
- Java servlets - ASP, .NET - PHP - Ruby - with CGI (all executeables, python, perl, etc.)	- JavaScript (and libraries - jquery, dojo, etc.) - Plugins (Flash, Java Applets, Silverlight, etc.)
Web frameworks	
- JSF - GWT	

3.5.2. Concept

The visualization framework provides a simple interface for human interaction. The user scenarios investigated within the requirement analysis are divided into modules. A module in this context is defined as a number of classes, which fulfill a certain purpose, related to human interaction (e.g. querying and displaying values of a certain datapoint). By dividing the architecture of the visualization into three levels (see Figure 29), coherent functionality can be grouped into reusable components (*ModuleControlle*, *DNDCtrl*, etc.).

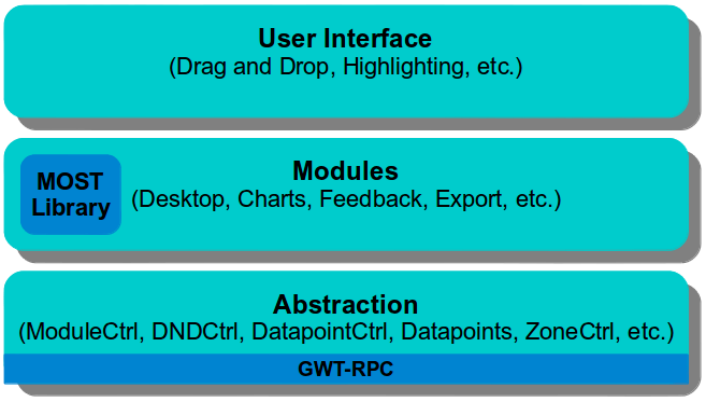


Figure 29. Software architecture of the proposed visualization framework

The abstraction layer encapsulates building information (*DatapointController*, *Datapoint*, etc.) and contains logic that cannot be directly manipulated by user input (e.g., *ModuleController* to manage the modules, *AuthenticationController* to manage security procedures, etc.). Communication to the server side is done with GWT-RPC requests. The *ModuleController* (see Figure 30) registers new modules (defined in the *ModuleRegistrator* class) and adds the main menu entry (*MainMenuEntry*) in the user interface. Figure 31 shows the main menu generated by the *ModuleController*.

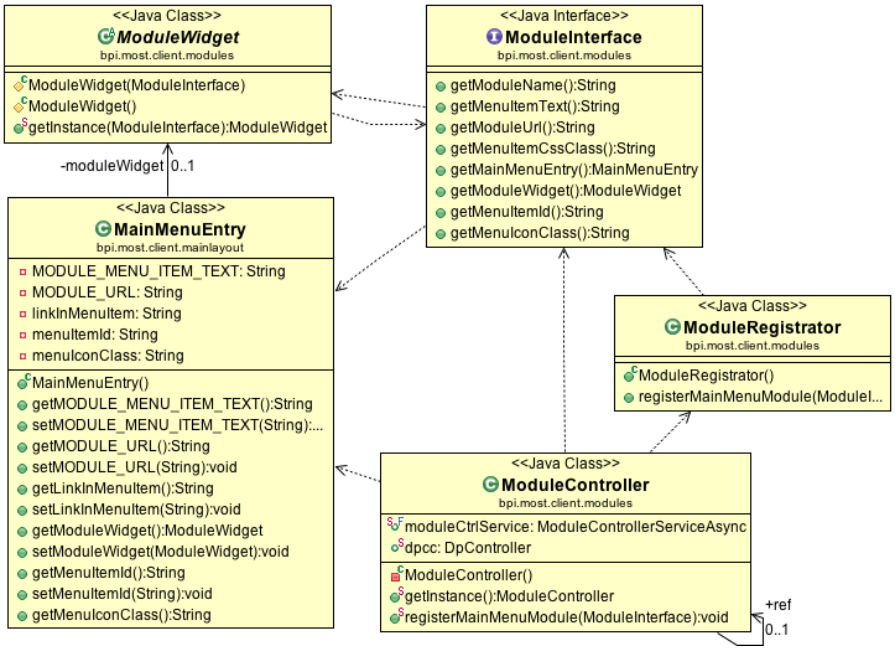


Figure 30. Methods provided by the Module Controller

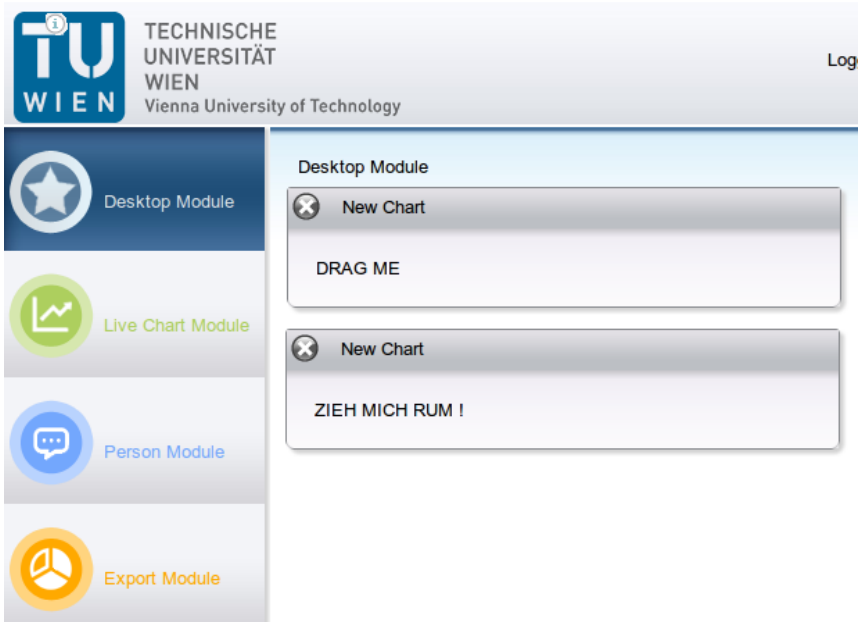


Figure 31. Main menu generated by the Module Controller

The module based design principles support the learning process of the user by emphasizing the logical modules visually. MOST provides a framework for developing and using custom modules that cover specific use cases. To add a new module to the webinterface, the *ModuleInterface* needs to be implemented and registered within the *ModuleRegistrator* class. The *ModuleInterface* contains only metadata of a module (module name, module URL, etc.). The respective module needs to implement the method *getModuleWidget()* which must return an instance of the actual graphical implementation of the module. This instance needs to inherit from the *ModuleWidget* class and contain all graphical components of the module. The module can be instantiated on demand, since metadata (*ModuleInterface*) and graphical components (*ModuleWidget*) are separated into different classes. This can improve the performance and the scalability of the overall webinterface. To offer a starting point in module development, a generic module implementation and various reusable components are available.

3.5.3. Visualization library

As shown in Figure 32, the visualization library provides various components (*MenuWidget*, *DragWidget*, etc.), which simplify module development and lead to a uniform design. The library is based on native GWT features and can be integrated in any GWT project.

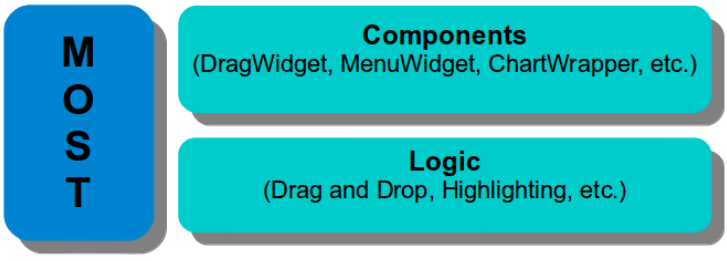


Figure 32. Visualization library of the proposed framework

An example is the class *DragWidget*. It can contain any kind of information, e.g., displaying a chart or presenting a formula as well as all generic GWT widgets (buttons, checkboxes, tables, etc.) (see GWT-Example 2012). Furthermore, the *DragWidget* supports the drag and drop logic of the proposed visualization framework. Detailed information and documentation about different visualization components can be found in the source-code (Javadoc) of MOST 2012.

3.5.4. Drag and Drop

By wrapping any custom information with a *DragWidget*, the object natively supports the basic drag and drop functionality within the user interface. The drag and drop (DND) features of GWT are enhanced with the *DNDController* (e.g. highlighting of droppable areas), as MOST uses drag and drop actions as the primary interaction method. By offering the user an alternative method to common interface interaction, MOST adds a dynamic feel to the user interaction process.

As shown in Figure 33, the DND logic communicates across all three software layers. To enable highlighting, a module must create a *DNDHighlightElement* object for each *DragWidget*. The *DNDHighlightElement* establishes a relation between the *DragWidget* and possible *DropWidgets* (which should be highlighted while dragging). During the drag process, the *DNDController* handles all DND related operations. The *DNDController* is part of the abstraction layer and is instantiated during the login process at the web interface. It subsequently listens for DND actions. The highlighting functionality is shown in Figure 34. When triggering a DND event, the *DNDController* highlights areas where a *DragWidget* is allowed to be dropped using the DOM-element of the widget.

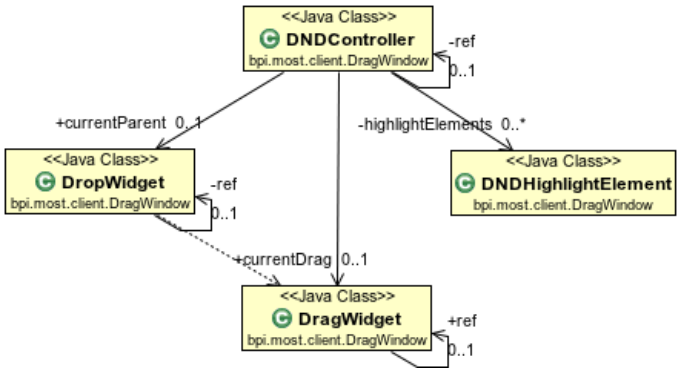


Figure 33. Relations of the DNDController within the client side software architecture

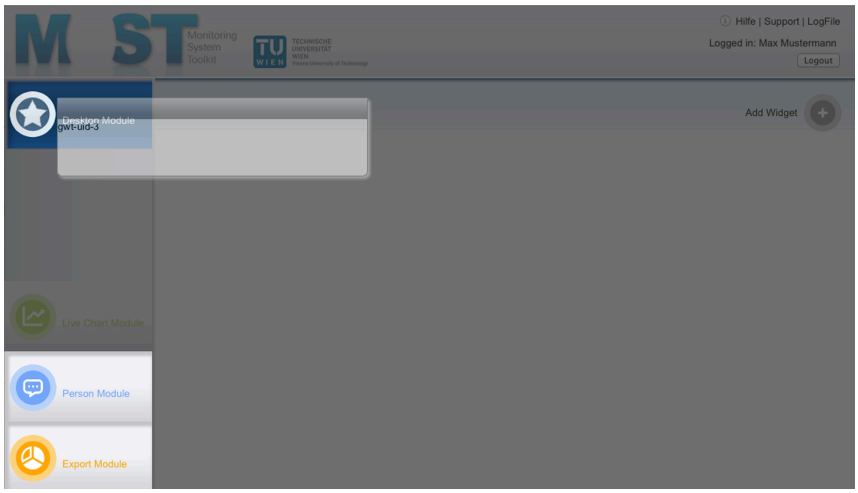


Figure 34. Highlighting of droppable areas while moving a DragWidget

3.5.5. Modules

To show the potential of the visualization framework, some modules covering various use cases were implemented.

Chart Module

The chart module enables creating trend charts from any datapoint by dropping it on a defined area in the module. Highcharts 2012 is used within a *DragWidget* as shown in Figure 35. To enable the visualization of timeframes containing a critical amount of measurements with the

highcharts library, the number of shown values needs to be reduced due to performance limits. Therefore, the data preprocessing methods - provided by the database and the virtual datapoints - are used to calculate a reduced number of values on behalf. Before drawing a chart, the amount of measurements in the requested timeframe is analyzed. If a critical amount of measurements is exceeded, the *getValuesPeriodic()* method is used to request a temporally set of data (periodic values) with a reduced amount of values. Finally, the preprocessed data is drawn in the chart. If the user zooms into the chart, new values are requested in the same way. This strategy reduces the high amount of measured data in a transparent manner. It shows how the data preprocessing functionality can be used to simplify application development.

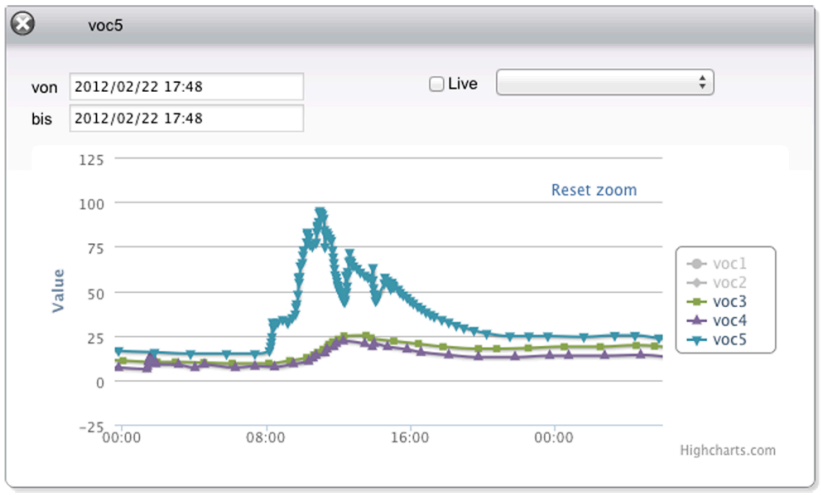


Figure 35. Data visualization using the chart module

Desktop Module

The desktop module enables the centralized visualization of information coming from different modules as shown in Figure 36. It allows moving (using drag and drop) any *DragWidget* to a central page (similar to the Microsoft Windows Desktop). This way, desired information from multiple sources can be combined on a single page.

Figure 36. Centralized data visualization using the desktop module

Three-dimensional building visualization

To map the location of datapoints within the real building, the modul “3D-Viewer” was developed. It visualizes a three-dimensional building model and respective datapoints based on the platform independent Web Graphics Library (WebGL). WebGL provides a 3d rendering engine supported by all current web browsers.

Prototypical implementations have been examined by adapting the projects IFCwebserver 2012 and BIMsurfer 2012. IFCwebserver 2012 uses the WebGL based SpiderGL 2012 library, supports the COLLADA 2012 file format, and showed various disadvantages during tests with building related models (performance issues, large model size, unstructured code, etc.). BIMsurfer 2012 extends the WebGL based SceneJS 2012 library, uses a JavaScript Object Notation (JSON) based file format and showed appropriate performance characteristics. Therefore, BIMsurfer 2012 was chosen to be enriched with building monitoring related features. An

example of the building visualization, using BIMsurfer 2012, is shown in Figure 37.

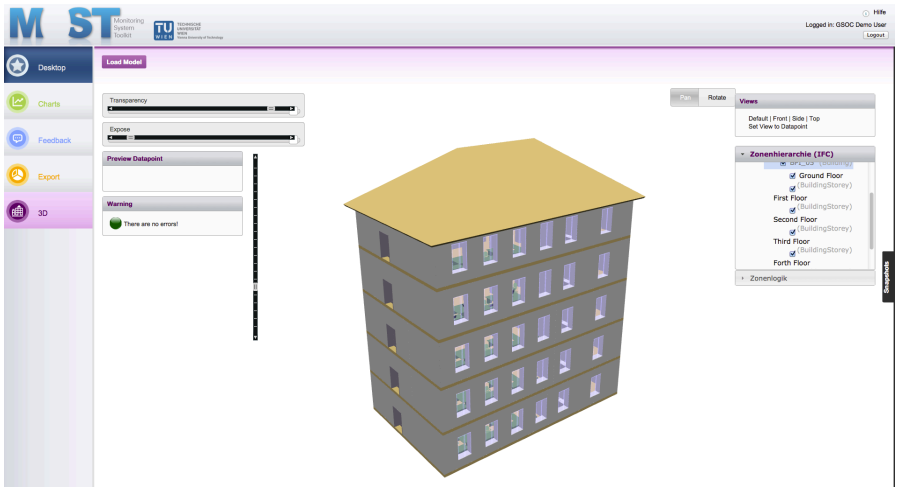


Figure 37. Three-dimensional building visualization using BIMsurfer 2012

The following use cases were investigated for the “3D-Viewer” module:

A. Lead the user within the three-dimensional building model to a desired datapoint and
B. show the user the location of a particular datapoint.

To support use case A in an intuitive way, additional building model browsing features were implemented. For example, diverse building stories can be exposed using a slider bar as shown in Figure 38. If no building story is marked, all levels are shifted. If a single floor is selected, only this one is exposed (see Figure 39). Furthermore, the transparency of the building walls can be controlled with an additional slider bar (see Figure 40).

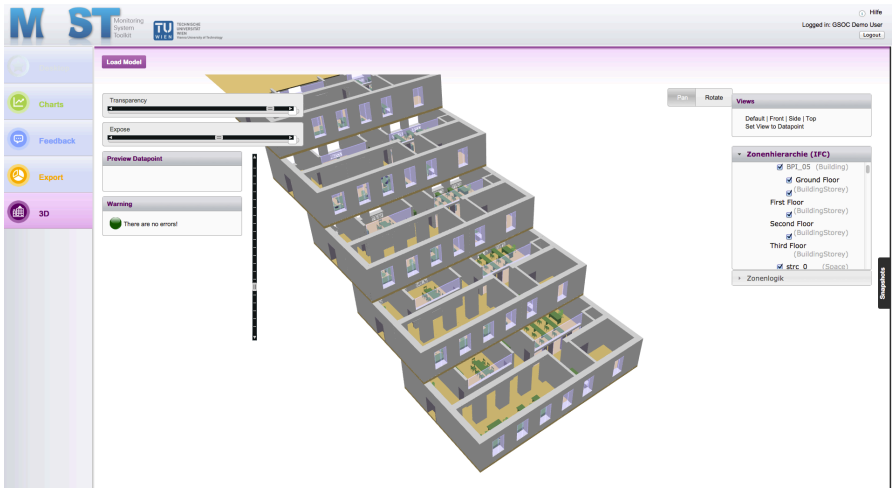


Figure 38. Exposing all levels of a building model

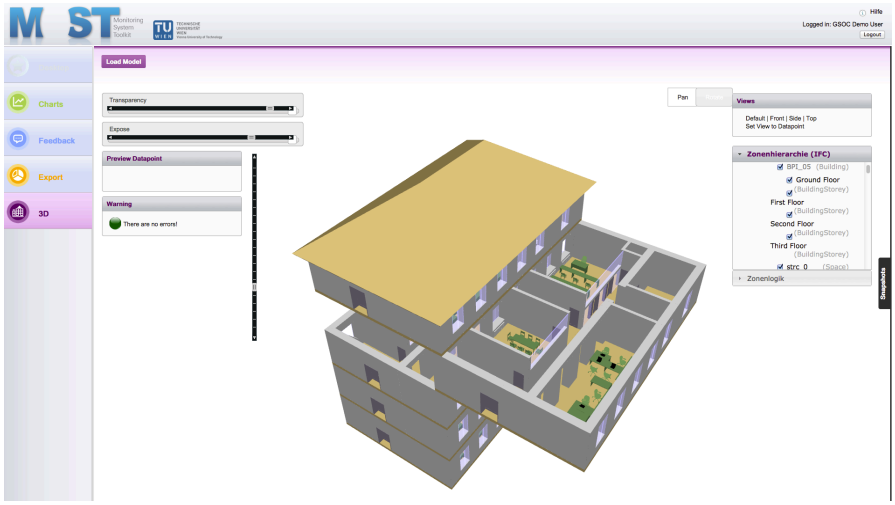


Figure 39. Exposing a single floor

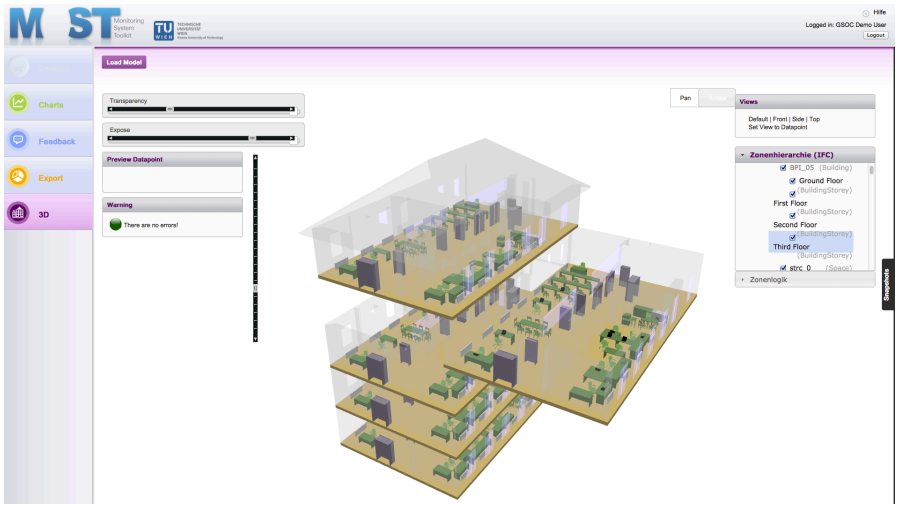


Figure 40. Building model visualization with enabled transparency

A preview of a particular datapoint (description, latest measurement, etc.) is shown by marking it via a single mouse click. To use the respective datapoint within any other module, drag and drop from the building model to the desired module is intended to be implemented.

Use case B (show the location of a particular datapoint) is planed to be supported by dynamically moving the view to the location of a requested datapoint. A prototypical implementation was tested based on the snapshot feature of BIMsurfer 2012. Future development intends to trigger the location visualization by dropping any datapoint object on the module.

To create a building model for the 3d viewer, any Computer-Aided Design (CAD) software, which supports the export of IFC2x4 2010 (Industry Foundation Classes) compatible files, can be used. Since the used CAD application did not support *IfcSensor* and *IfcActor* objects, a naming convention based approach was used to link IFC objects to respective datapoints. Any IFC object using the syntax “dp_<datapoint name>” is processed as a datapoint in the 3d viewer (preview on click, drag and drop support, etc.). To convert the IFC file to the required JSON format, BIMserver 2012 is used.

Module development

To implement further modules, generic sample code is provided. An export module is intended to be implemented to show how data can be requested based on various rules (e.g., only workdays, only defined time slots, etc.). Javadoc 2012 is used to document the developed source code. Figure 41 gives an overview of currently implemented packages and classes of the visualization framework. Online access to the documentation is available at http://most.bpi.tuwien.ac.at/doc/javadoc-client/index.html

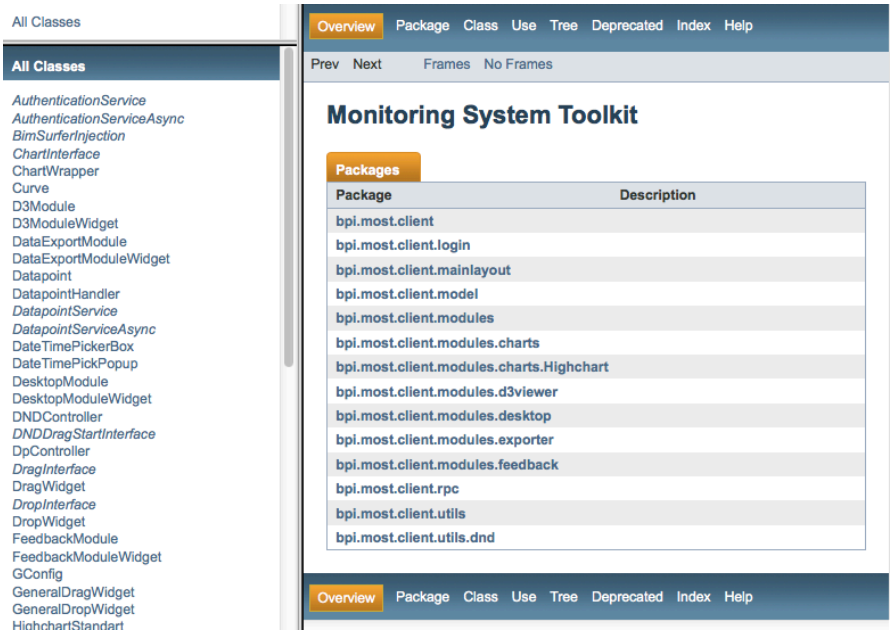


Figure 41. Source code documentation of the visualization framework

Chapter 4.
Prototypical implementation

To study real-life implementation and application scenarios, two buildings were selected and partly equipped with necessary monitoring infrastructure. These buildings house offices, labs and lecture rooms of the Vienna University of Technology.

One of the buildings (Lehartrakt, see Figure 42) was completed 2010 and provides reusable building automation infrastructure to various degrees. The second building (Mitteltrakt, see Figure 43) was built more than 100 years ago and provides no reusable building automation infrastructure. As such, these buildings are representative of a large number of existing buildings in Vienna. They thus represent a wide range of technical challenges that need to be met in order to realize the postulated dynamic data acquisition and processing architecture in the context of existing buildings. Such challenges pertain specifically to the technology update requirements for incorporation of high-resolution sensory and metering capabilities, device connectivity, and cross-platform data transfer.

Figure 42. Building Lehartrakt

Figure 43. Building Mitteltrakt

4.1. Lehartrakt

Recently completed (2010), Lehartrakt is equipped to various degrees with current building automation technologies. Therefore, the monitoring system can reuse some of the sensor and network infrastructure to reduce installation efforts. Figure 44 shows a four layer model of the entire monitoring infrastructure.

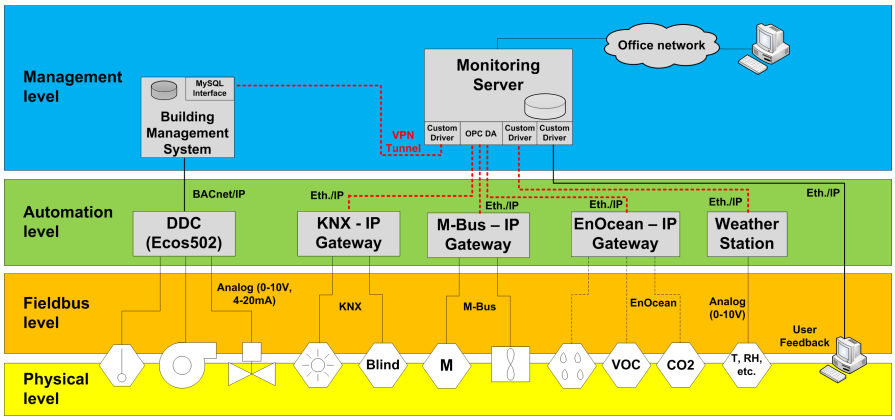


Figure 44. Four layer model of the monitoring system in Lehartrakt

Many sensors in the existing BACnet, KNX, and M-Bus networks are reused. Electricity meters are added to the M-Bus system. All other sensors are added with the wireless fieldbus EnOcean to reduce installation costs. KNX, M-Bus and EnOcean networks are accessed by using the OPC DA Connector. Heating ventilation and air conditioning (HVAC) is controlled by a proprietary Building Management System (BMS) which provides an Open Database Connectivity (ODBC) interface only. Therefore the JDBC Connector is used to poll required data.

The monitored area in Lehartrakt covers two labs, two office rooms and one conference room. Figure 45 and Figure 46 show the floor plans including all sensors. Energy use of hydronic radiators is monitored with heat meters, which measure incoming and outgoing water temperature and the volume flow (see Figure 47). The energy use of a fan-coil units is calculated by measuring the temperature difference between the incoming and outgoing air and the air volume flow (see Figure 48). Relative humidity, room air temperature, window/door states, and occupancy are monitored with self-powered sensors using the wireless fieldbus EnOcean (see Figure 49 to Figure 51). Carbon dioxide and volatile organic components are measured with legacy sensors, which are equipped with EnOcean wireless modules. Electrical energy use is measured with M-Bus meters. Light and blind states are monitored by tapping the KNX fieldbus. Diverse gateways are used to access all fieldbus technologies based on Ethernet/IP. To enables secure remote data access, the Ethernet/IP network is tunneled to the monitoring server using a VPN concentrator. A Linksys WRT54GL router with an OpenWRT 2012 installation is used to establish the VPN connection. OpenVPN 2012 is used as VPN technology. A watchdog scripts on the router periodically probes the VPN connection and initiates a reconnection if communication problems occur. The installation setup of the EnOcean and the M-Bus Gateway including a router to establish the VPN connection is shown in Figure 52.

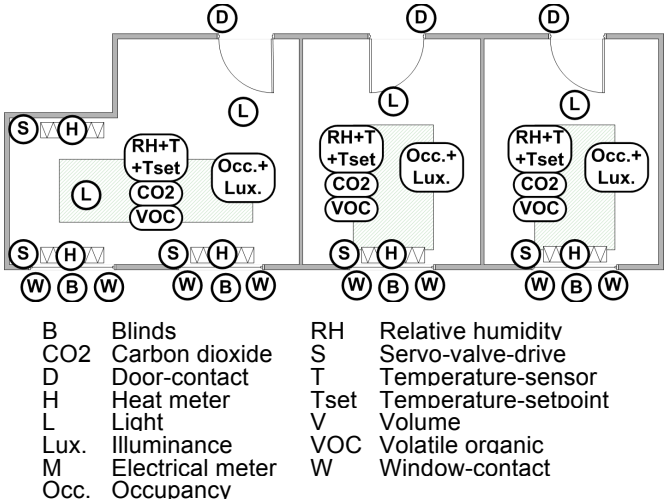


Figure 45. Floor plan including sensors of two office rooms and one conference room in Lehartrakt

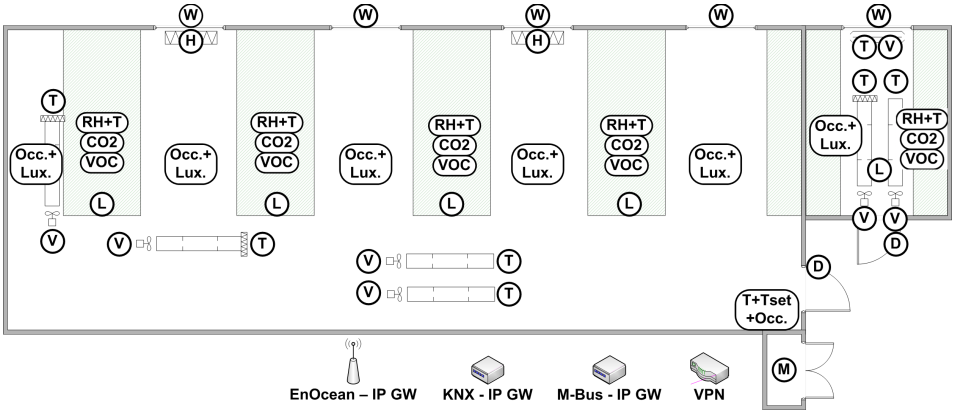


Figure 46. Floor plan including sensors of two laboratories in Lehartrakt

Figure 47. Heat meters installed at radiators

Figure 48. Sensor setup at the fan-coils

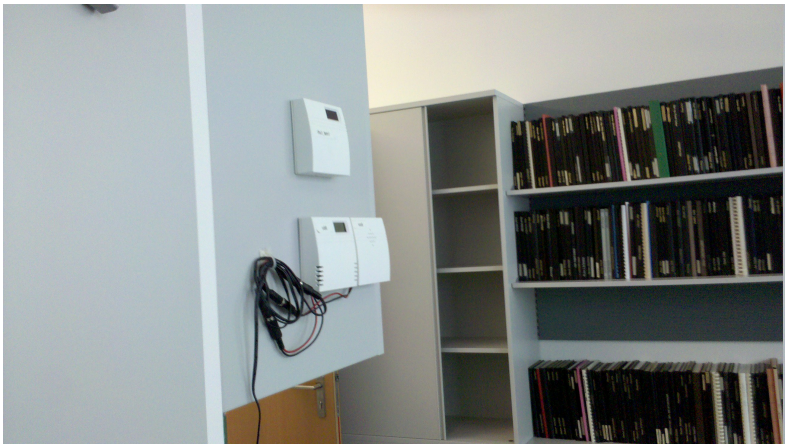


Figure 49. Temperature, relative humidity, CO2 and VOC sensor setup

Figure 50. Occupancy detection based on a wireless motion sensor

Figure 51. Contact sensor used to measure window state

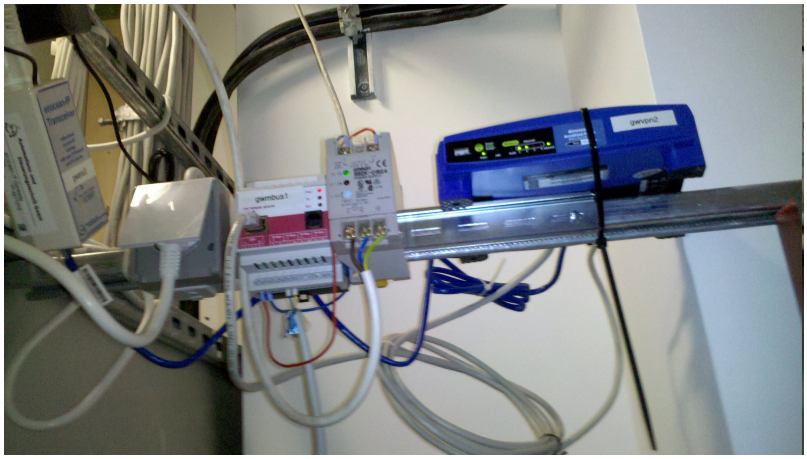


Figure 52. Installation setup of various fieldbus gateways

4.2. Mitteltrakt

The building Karlsplatz provides no reusable building automation infrastructure at all. To reduce installation costs, a fully independent wireless approach for the fieldbus network is used. Figure 53 shows the four layer model of the entire monitoring infrastructure in Mitteltrakt. To transfer the measured data from the EnOcean - IP gateway to the monitoring server, a VPN-concentrator is used.

EnOcean was chosen as the wireless fieldbus system because of its optimized design for low power use. It allows the construction of self-powered sensor devices, which reduces installation efforts and increases installation flexibility. Nevertheless, some sensors still need a power supply due to the energy use of the sensor technology. For example, a CO2 sensor needs a significant power supply to drive its heating coil. Many sensors with integrated EnOcean communication technology are available on the market. Missing sensors can be easily developed using legacy devices and EnOcean-modules of the STM and TCM series as shown in Figure 54.

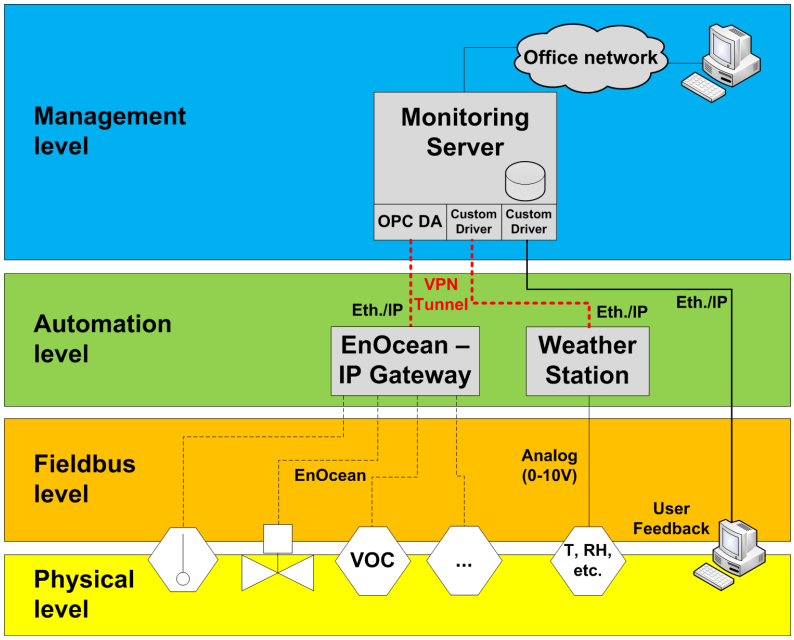


Figure 53. Four layer model of the monitoring system in Mitteltrakt

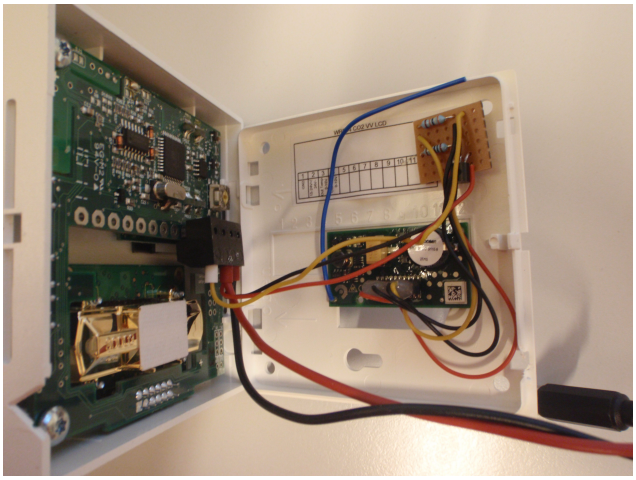


Figure 54. Legacy CO2 sensor equipped with the EnOcean wireless module STM110

Figure 55 shows the floor plan of the monitored area including all sensors. The heating energy transfer rate of the radiators is calculated by measuring the temperature of the radiator and the room (see Figure 56). Using the K values described in DIN 1994, heating energy use can be estimated. States of lights are detected by measuring their electrical

energy use (see Figure 57). A small rack is used to avoid unintended covering of air quality sensors (CO2, VOC, temperature, relative humidity) and to measure close to the real working area (see Figure 58). Occupancy detection for single work areas is realized by limiting the range of motion sensors to the desired place (see Figure 59).

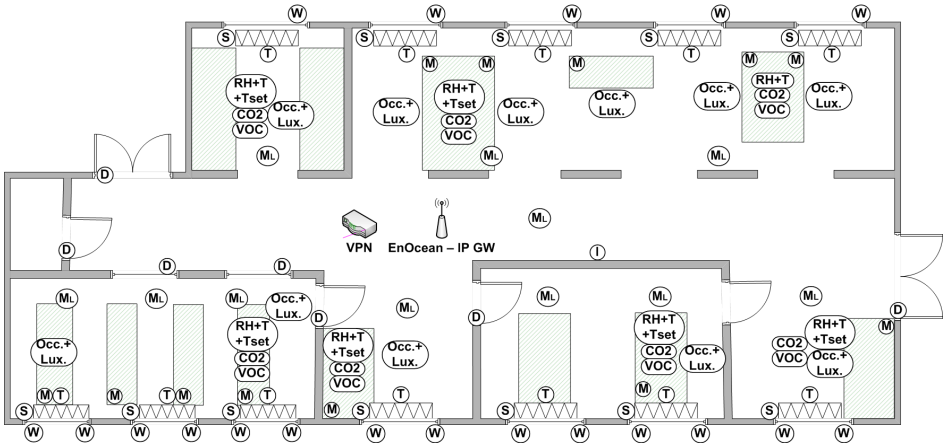


Figure 55. Floor plan including sensors in Karlsplatz

Figure 56. Sensor setup to measure the mean radiator temperature

Figure 57. Electrical meter installed on lights

Figure 58. Rack for air quality sensors

Figure 59. Occupancy detection using motion sensors

4.3. IT infrastructure

Measurements of both monitoring implementations are processed within the same Information Technology (IT) infrastructure. To improve scalability, flexibility and stability, virtual machines are used. Diverse components of the proposed toolkit are installed in respective virtual machines. VirtualBox 2012 is used as virtualization technology. Different virtual machines are used for the

- Connector
  Virtual machine based on Microsoft Windows.
- VPN-concentrator
  OpenVPN installed within a Debian/Linux system.
- Database
  MySQL installed within a Debian/Linux system.
- Data-abstraction framework including the GWT based web-interface
  Apache Tomcat/TomEE Java Application Server supporting the Java Enterprise Edition installed within a Debian/Linux system.
- a number of virtual machines for diverse processing applications
  MATLAB, EnergyPlus, etc. is installed within respective operating systems.

Figure 60 shows the webinterface of the server system running all productive virtual machines. It provides 12 GByte of memory and eight Intel Xeon CPU cores. To enable stress tests, applied during development, an additional virtual machine server, providing 24 GByte of memory and twelve Intel Xeon CPU, is used. Furthermore, an independent computer is used to automatically backup all virtual machines and databases.

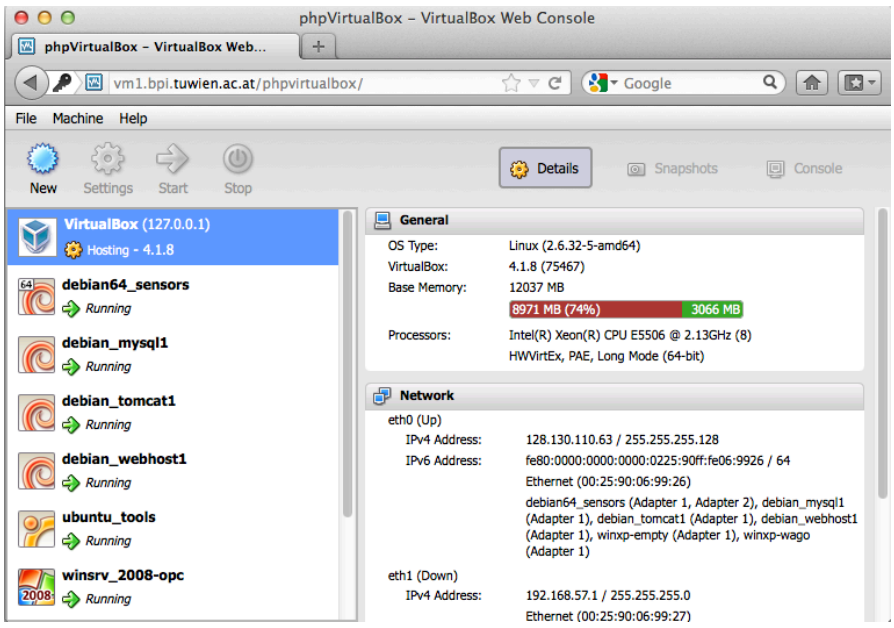


Figure 60. Webinterface of the productive virtual machine server

4.4. Data utilization

The measurements resulting from the described monitoring implementation can be applied to various ends (fault detection, improving building operation, etc.). This section shows only a few application examples.

4.4.1. Increase user awareness

To increase the awareness of building users regarding their impact on buildings' energy use, an information screen, supporting touch interaction, is installed in the Department of Building Physics of the Vienna University of Technology. Figure 61 shows demonstratical usage.

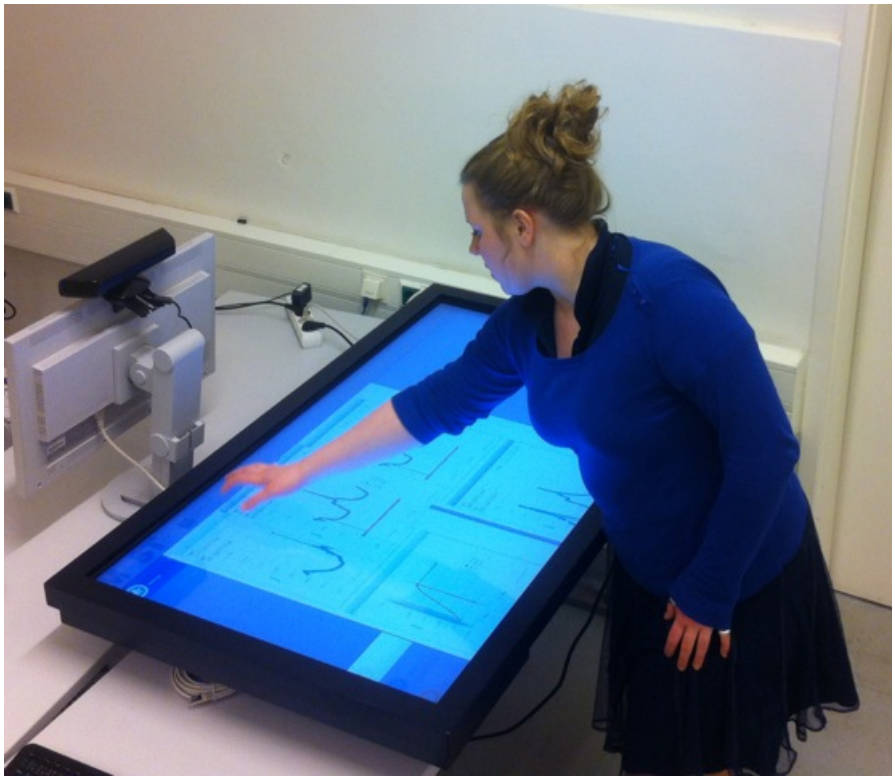


Figure 61. User interaction with the webinterface using touch control

4.4.2. Simulation model calibration

In the context of building performance simulation one of the key problems is to determine the accuracy and reliability of simulation codes and tools (Mahdavi 2001, Mahdavi et al. 2009). Certain core algorithms in building performance simulation codes may be validated analytically, with major simplifications of domain complexity and boundary condition assumptions. Fundamental physical processes mapped in building performance codes may be also tested via tightly controlled experiments in appropriately designed test spaces and facilities. But real buildings are far more complex, given contingencies in terms of occupancy processes and use patterns, weather conditions, as well as initial and emerging uncertainties in semantic properties assumptions. Such analytical validation techniques or test in highly controlled conditions are hardly applicable to complex real buildings. Comprehensive monitored data from

buildings, on the other hand, can be advantageously deployed to obtain a reliable empirical basis for the evaluation and calibration of respective simulation models.

In this context the monitored data of the building Mitteltrakt was used to (Tahmasebi and Mahdavi 2012):

- i) create a weather file based on local data instead of using a predefined "typical" year,
- ii) populate the initial building model with dynamic data regarding internal loads, device states, and occupancy processes,
- iii) to calibrate the initial model.

Table 12 shows all data streams used during the calibration process.

Table 12. Use of monitoring data for building simulation calibration (Tahmasebi and Mahdavi 2012)

Use of data	Datapoint	Unit
Creating local weather data file	Global horizontal radiation	[$W.m^{-2}$]
	Diffuse horizontal radiation	[$W.m^{-2}$]
	Outdoor air dry bulb temperature	[^{o}C]
	Outdoor air relative humidity	[%]
	Wind Speed	[$m.s^{-1}$]
	Wind direction	[degree]
	Atmospheric pressure	[Pa]
Creating the initial model	Electrical plug loads	[W]
	State of windows and doors (open/closed)	[-]
	State of the lights (on/off)	[-]
	Occupancy (presence/absence)	[-]
Calibrating the model	Indoor air dry bulb temperature	[^{o}C]

The building was modeled within EnergyPlus 2012. The monitored data were incorporated into EnergyPlus by defining schedules. The schedule

files are generated using a MATLAB script. A singe file was created for each datapoint in an event-based manner (compact schedule). To calibrate the desired simulation model, the optimization tool GenOpt 2012 was used. Data obtained via the monitoring system were deployed to both populate the initial simulation model and to maintain its fidelity through a systematic optimization-assisted calibration process. The results showed noticeable improvement of the predictive potency of the calibrated model. Figure 62 and Figure 63 compare monitored office temperature with initial and calibrated model results, during a 9-day period of calibration and validation. Details about the calibration process are described in Tahmasebi and Mahdavi 2012.

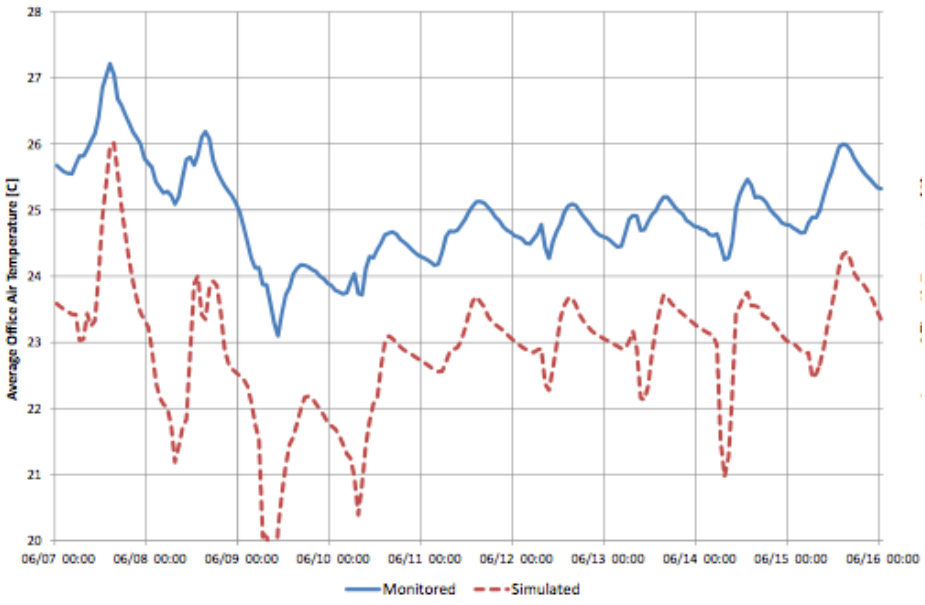


Figure 62. Monitored & simulated office temperature - initial model (Tahmasebi and Mahdavi 2012)

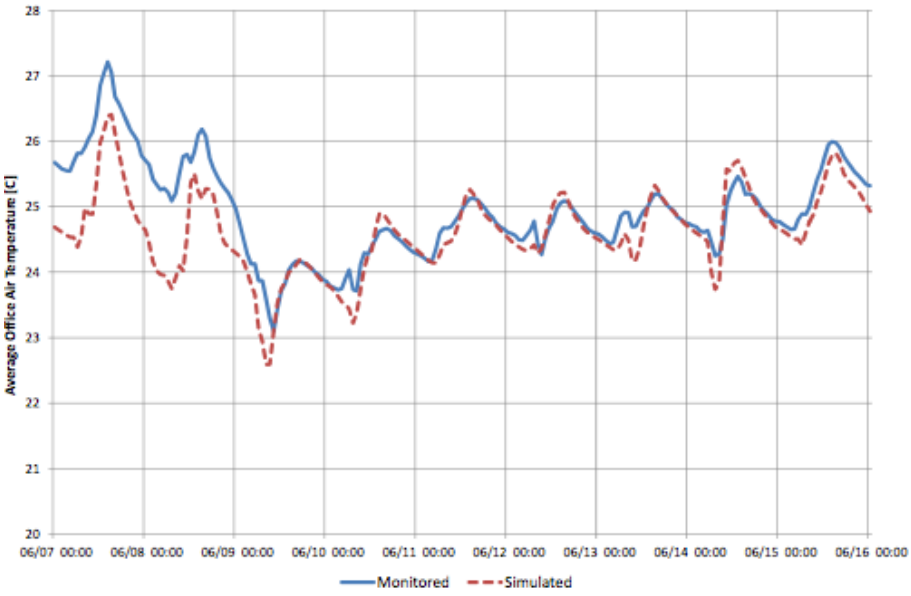


Figure 63. Monitored & simulated office temperature - calibrated model (Tahmasebi and Mahdavi 2012)

Chapter 5.
Conclusion

5.1. Contributions

By using the proposed toolkit, multiple salient data streams originating from a building's operation can be analyzed in a comprehensive manner. Thus, the envisioned use cases can be realized. Provided interfaces enable real-time data processing independent of software. Powerful data preprocessing methods are implemented that support effective data analysis. Such methods involve, for example, linked queries of energy use for specific time intervals and building zones under specific occupancy conditions. The scalable design of the monitoring framework can accommodate very different system requirements and supports new buildings (with existing building automation components) as well as independently conceived and implemented monitoring systems that can be realized in buildings without reusable infrastructure. The framework developed and the associated applications support building operators to rapidly respond to occupants' requirements. The toolkit is shown to be flexible, as demonstrated via successful implementations in two very distinct reference buildings.

The proposed database design showed how multiple building data streams can be processed on a long-term basis. Powerful data preprocessing algorithms allow effective data analysis and support realizing the envisioned use cases. Various performance optimizations in the database design minimize required computer resources. Practical implementation demonstrated the usability of the database. The benchmark tests showed a practical limit of about 900.000 datapoints delivering about 4 measurements per hour. Two billion measurements produce a database size of 160 GByte with the current design. This is equivalent to about 110.000 datapoints storing a new value every 30 minutes for a period of one year. The high performance of the preprocessing functions implemented in the database enables the

generation of data sets (with a desired structure) in real-time. The research efforts showed the limits of using relational databases for storing diverse building data streams. Since common relational databases do not support an adequate separation of measurements in multiple (independently indexed) dimensions (e.g. MySQL partitions can only split up values based on one criteria), an inefficient memory usage may result. For example, in the proposed database design, all datapoints within one partition (timeframe) are cached (indexed) the same way, even if they are never used. Reduced memory usage could be achieved by keeping only the currently relevant datapoints and timeframes in memory. This caching strategy could be implemented by realizing the long term data storage (currently covered with the data table) with an appropriate NoSQL solution. Due to the cheap memory costs of present computers, this limitation is only relevant for very large implementations.

The data-abstraction framework provides desired information in a generic manner. By using virtual datapoints, multiple measurement sources can be incorporated into a single data stream. Furthermore, complex data gathering (e.g. calculation of values using simulation tools) can be abstracted behind a simple interface (datapoint). By providing a web-based visualization framework, user interface development can be significantly simplified. Thus, increased awareness of building users and improved BMS control can be fulfilled in a platform independent manner.

The overall toolkit consists of more then 100.000 lines of code with additional 30.000 lines of documentation. More then 1.400 subversion commits containing about 12.500 changes have been integrated. Information on further developments is available at http://most.bpi.tuwien.ac.at.

5.2. Future research

Future research and development challenges involve the implementation of a software independent OPC DA connector to extend supported

building systems, improvement of the OPC UA interface to increase compatibility with processing applications, enhancement of the web visualization to simplify usage for non-technical skilled users, and demonstration of various use cases (e.g. automatic calibration of simulation models, simulation-based fault detection, remote data collection for a number of projects, etc.).

To further improve historical building data storage, future research challenges involve reduction of the database size by optimizing the storage engine (e.g. switching to MyISAM), developing stored procedures to automatically resort measurements in appropriate MySQL partitions, performance analyses of the watchdog algorithm, and performance optimization by replacing the datapoint name (varchar) of the data table with an integer identification number.

Currently, the following projects are treated within the Google Summer of Code (GSOC 2012).

5.2.1. Building viewer

Missing features of the proposed three-dimensional building viewer will be implemented. Afterwards, the usability using diverse user interface interaction technologies (mouse/keyboard, touch and gestures/natural interaction) will be analyzed.

5.2.2. OPC connector

Since the existing OPC connector has limited adaption options, caused by the proprietary software environment, an open-source alternative will be implemented. The proposed implementation will extend the connector framework and use JEasyOPC 2012 to access OPC data sources. A simple user interface is intended to be implemented on top.

5.2.3. Data exporter

Using the GWT based web-visualization framework, a building data exporter will be implemented. The exporter will be able to create different file formats (CVS, etc.) based on various rules (only workdays, when zone is occupied, etc.). For example: export hourly temperature values, when zone XXX is occupied, from the 01.04.2011 to the 01.06.2011, on workdays only.

5.2.4. OPC UA server

Based on the data-abstraction framework, an OPC UA server interface will be implemented. Supporting OPC UA enables various processing applications to easily access building data in real-time. The required OPC UA information model will be based on datapoint and zone definitions. Thus, the OPC UA tree (resulting from the information model) is build based on zones with datapoints as lead nodes. Methods of the leaf nodes (datapoints) enable data access.

5.3. Publications

As of this writing, various portions and reports on earlier stages of this work have been published in the following articles:

R. Zach, M. Schuss, R. Bräuer and A. Mahdavi. *Improving building monitoring using a data preprocessing storage engine based on MySQL.* 25 – 27 July, European Conference of Product and Process Modelling (ECPPM 2012), Reykjavik, Island

R. Zach, S. Glawischnig, M. Hönisch, R. Appel, A. Mahdavi. *MOST: An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization.* 25 – 27 July, European Conference of Product and Process Modelling (ECPPM 2012), Reykjavik, Island

R. Zach, S. Glawischnig, R. Appel, J. Weber, A. Mahdavi. *Building data visualization using the open-source MOST framework and the Google*

Web Toolkit. 25 – 27 July, European Conference of Product and Process Modelling (ECPPM 2012), Reykjavik, Island

R. Zach, A. Mahdavi: "*MOST - Designing a vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization*"; Vortrag: 1ICAUD, Tirana, Albanien; 18.04.2012 - 21.04.2012; in: "*Proceedings - First International Conference on Architecture and Urban Design - 1-ICAUD*", EPOKA Univ.; Dep. of Arch. (Hrg.); Epoka University Press, 1 (2012), ISBN: 9789928-135-01-8; 7 S.

R. Zach, R. Bräuer, M. Schuss, A. Mahdavi: "*A Monitoring Framework for Runtime Simulation Calibration and Validation*"; Vortrag: Building Simulation 2011 - IBPSA 2011, Sydney, Australien; 14.11.2011 - 16.11.2011; in: "*driving better design through simulation - Proceedings of the 12th Conference of The International Building Performance Simulation Association*", V. Soebarto, H. Bennetts, P. Bannister, P.C. Thomas, D. Leach (Hrg.); Konferenzpublikation mit wissenschaftlichem Lektorat, (2011), ISBN: 978-0-646-56510-1; S. 926 - 932.

R. Zach, M. Schuss, C. Pröglhöf, K. Orehounig, R. Bräuer, A. Mahdavi. *An integrated architecture for energy systems and indoor climate monitoring in buildings*. 16. – 18. Februar, Vienna, 7. Internationale Energiewirtschaftstagung (2011).

M. Schuss, R. Zach, K. Orehounig, A. Mahdavi: "*Empirical Evaluation of a predictive Simulation-Based Control Method*"; Vortrag: Building Simulation 2011 - IBPSA 2011, Sydney, Australien; 14.11.2011 - 16.11.2011; in: "*driving better design through simulation - Proceedings of the 12th Conference of The International Building Performance Simulation Association*", V. Soebarto, H. Bennetts, P. Bannister, P.C. Thomas, D. Leach (Hrg.); Konferenzpublikation mit wissenschaftlichem Lektorat, (2011), ISBN: 978-0-646-56510-1; S. 918 - 925.

S. Chien, R. Zach, A. Mahdavi: "*Developing user interfaces for monitoring systems in buildings*"; in: "*Proceedings of the IADIS International*

Conference - Interfaces and Human Computer Interaction 2011", K. Blashki (Hrg.); iadis - international association for development of the information society, Rome, 2011, ISBN: 978-989-8533-00-5, Paper-Nr. 4, 8 S.

R. Zach, A. Mahdavi: "*Monitoring for Simulation Validation*"; Vortrag: BauSim2010 - Building Performance Simulation in a Changing Environment, Technische Universität Wien; 22.09.2010 - 24.09.2010; in: "*BauSim 2010 - Building Performance Simulation in a Changing Environment*", A. Mahdavi, B. Martens (Hrg.); Verlag / Organisation / Universität mit wissenschaftlichem Lektorat, Wien (2010), ISBN: 978-3-85437-317-9; S. 190 - 195.

Chapter 6.
References

6.1. Literature

ASHRAE 2004. ANSI/ASHRAE, Std. 135, BACnet – *A Data Communication Protocal for Building Automation and Control Networks*

BIMserver 2012. Open source Building Information Modelserver, June 2012, *http://bimserver.org*

BIMsurfer 2012. Webviewer of IFC/BIM models based on WebGL, February 2012, *http://bimsurfer.org*

Cassia F. 2007. Open Source, the only weapon against "planned obsolescence", June 2012, *http://www.theinquirer.net/inquirer/news/1001739/open-source-weapon-planned-obsolescence/*

Chien S., Zach R., Mahdavi A. 2011. *Developing user interfaces for monitoring systems in buildings*. Proceedings of the IADIS International Conference - Interfaces and Human Computer Interaction 2011. Rome, 2011, ISBN: 978-989-8533-00-5, Paper-Nr. 4, 8 S., pp 29 - 36

COLLADA 2012. XML schema that enables digital asset exchange within 3d models. June 2012, *https://collada.org*

Daniels K. 2003, *Advanced Building Systems, A Technical Guide for Architects and Engineers*, Birkhäuser

DCOM 2012. Distributed Component Object Model, June 2012, *http://msdn.microsoft.com/library/cc201989.aspx*

DIN 1994. DIN EN 834, *Heat cost allocators for the determination of the consumption of room heating radiators - Appliances with electrical energy supply*

EnergyPlus 2012. EnergyPlus Energy Simulation Software, June 2012, *http://apps1.eere.energy.gov/buildings/energyplus/*

GenOpt 2012. Optimization program for the minimization of a cost function, June 2012, *http://simulationresearch.lbl.gov/GO/*

Gökce H.U. 2010. *Multi Dimensional Analysis of Building Performance Data for Energy Efficient Building Operation*. Dissertation. University College Cork

Graubner C. and Hüske K. 2003. *Nachhaltigkeit im Bauwesen*. Ernst & Sohn. ISBN 3-433-01512-0.

GSOC 2012. Google Summer of Code, February 2012, *http://code.google.com/soc/*

GWT 2012. Google Web Toolkit, February 2012, *http://code.google.com/webtoolkit/*

GWT-Example 2012. Showcase of GWT widgets, February 2012*, http://gwt.google.com/samples/Showcase/*

Highcharts 2012. Interactive JavaScript charts for web projects, February 2012, *http://www.highcharts.com*

IFC2x4 2010. *Industry Foundation Classes,* May 2011, *http://buildingsmart-tech.org/ifc/IFC2x4/rc2/html/index.htm*

IFCwebserver 2011. Webviewer of IFC/BIM models, February 2012, *http://code.google.com/p/ifcwebserver/*

IPCC 2009. *http://www.ipcc.ch/ipccreports/index.htm*, January 2012

ISO 2004. ISO 16484-2, *Building automation and control systems (BACS)* – Part 2: Hardware

Iwanitz F. and Lange J. 2002. *OPC: Grundlage, Implementierung und Anwendung*. Heidelberg: Hüthig. ISBN 3-7785-2866-1

Javadoc 2012. Tool to generate documentation, June 2012, *http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html*

JEasyOPC 2012. Java based OPC DA Client, February 2012, *http://jeasyopc.sourceforge.net*

Kastner W., Neugschwandtner G., Soucek S., Newman H.M. 2005. *Communication systems for building automation and control*, Proceedings of IEEE, vol. 93, no. 6, pp. 1178-1203

Kemper A. and Eickler A. 2009. *Datenbanksysteme*. 7. Auflage. Oldenbourg Wissenschaftsverlag GmbH, ISBN: 978-3-486-59018-0

KNX 2004. Konnex Association, *KNX Specifications*, Version 1.1

License 2012. Creative Commons Attribution-ShareAlike 3.0 Unported License, June 2012, *http://creativecommons.org/licenses/by-sa/3.0/*

LON 2010. LonMark International, June 2010, *http://www.lonmark.org*

Mahnke W., Leitner S.H., Damm M. 2009. *OPC Unified Architecure*. Springer-Verlag Berlin Heidelberg. ISBN 978-3-540-68898-3

Mahdavi A. 1997. *Toward a Simulation-assisted Dynamic Building Control Strategy*. Proceedings of the Fifth International IBPSA (International Building Performance Simulation Association) Conference. Vol. I, pp. 291 - 294.

Mahdavi A. 2001. *Simulation-based control of building systems operation.* Building and Environment. Volume 36, Issue 6, ISSN: 0360-1323. pp. 789-796.

Mahdavi A., Mohammadi A., Kabir E., Lambeva L. 2008. *Shading and Lighting Operation in office Buildings in Austria: A Study of User Control Behavior;* Building Simulation, Volume 1 Number 2 June 2008 (2007), S. 111 - 117.

Mahdavi A., Schuss M., Suter G., Metzger S., Camara S., Dervishi S. 2009. *Recent advantages in simulation-powered building systems control*. July 27 – 30, Glasgow, Scotland, Eleventh International IBPSA Conference.

Mahdavi A. 2009. *Patterns and Implications of User Control Actions in Buildings*. Journal Indoor and Built Environment, 18, 5; S. 440 - 446.

Mahdavi A., Orehounig K., Pröglhöf C. 2009. *A simulation-supported control scheme for natural ventilation in buildings*. Proceedings of the 11th IBPSA Conference, Building Simulation 2009, Glasgow, Scotland, pp. 783 - 788.

MOST 2012. *Monitoring System Toolkit*, February 2012, *http://most.bpi.tuwien.ac.at*

MySQL 2012. *Multi-column indexes to improve database performance*, February 2012, *http://dev.mysql.com/doc/refman/5.1/en/multiple-column-indexes.html*

Neumann C. and Jacob D. 2008. *Guidelines for the evaluation of building performance*. Freiburg, Germany: Fraunhofer Institute for Solar Energy Systems.

OASIS 2006. *oBIX 1.0* Committee Specification 01, June 2012, *http://www.obix.org*

O'Donnell J. 2009. *Specification of Optimum Holistic Building Environmental and Energy Performance Information to Support Informed Decision Making*. Doctorate, University College Cork, Ireland.

OPC 2012. OPC Foundation, June 2010, *http://www.opcfoundation.org*

OPC Datahub 2012. OPC Toolkit, February 2012, *http://www.opcdatahub.com*

OpenWRT 2012. A Linux distribution for embedded devices, June 2012, *https://openwrt.org*

OpenVPN 2012. Open source SSL VPN solution, June 2012, *https://openvpn.net*

Orehounig K., Mahdavi A., Pröglhöf C., Schuss M. 2010. *Virtual implementation of a simulation-assisted passive cooling strategy in*

buildings, 9 – 12 May, Antalya, Clima 2010, International Conference on Sustainable Energy Use in Buildings.

Raftery P., Keane M., O'Donnell J., Costa A. 2010. *Energy Monitoring Systems: value, issues and recommendations based on five case studies*. 9 – 12 May, Antalya, Clima 2010, International Conference on Sustainable Energy Use in Buildings.

SceneJS 2012. JSON-based scene graph API for WebGL, June 2012, *http://scenejs.org*

SpideGL 2012. JavaScript 3D Graphics library that relies on WebGL, June 2012, *http://spidergl.org*

Statistik Austria 2007. Bestand an Gebäuden und Wohnungen, June 2012, *http://www.statistik.at/web_de/statistiken/wohnen_und_gebaeude/bestand_an_gebaeuden_und_wohnungen/index.html*

Tahmasebi F. and Mahdavi A. 2012. *Monitoring-based optimization-assisted calibration of the thermal performance model of an office building*; Vortrag: 1ICAUD, Tirana, Albanien; 18.04.2012 - 21.04.2012; in: "*Proceedings - First International Conference on Architecture and Urban Design - 1-ICAUD*", EPOKA Univ.; Dep. of Arch. (Hrg.); Epoka University Press, 1 (2012), ISBN: 9789928-135-01-8; 5 S.

Tomcat 2012. The tomcat JDBC connection pool, March 2012, *http://people.apache.org/~fhanik/jdbc-pool/jdbc-pool.html*

VirtualBox 2012. Open-Source x86 virtualization, June 2012, *https://www.virtualbox.org*

6.2. Tables

6.3. Figures

Chapter 7.
Appendix

7.1. MySQL Stored Procedures

The following figures show the flow diagrams of the data preprocessing algorithms implemented in the MySQL database using Stored Procedures.

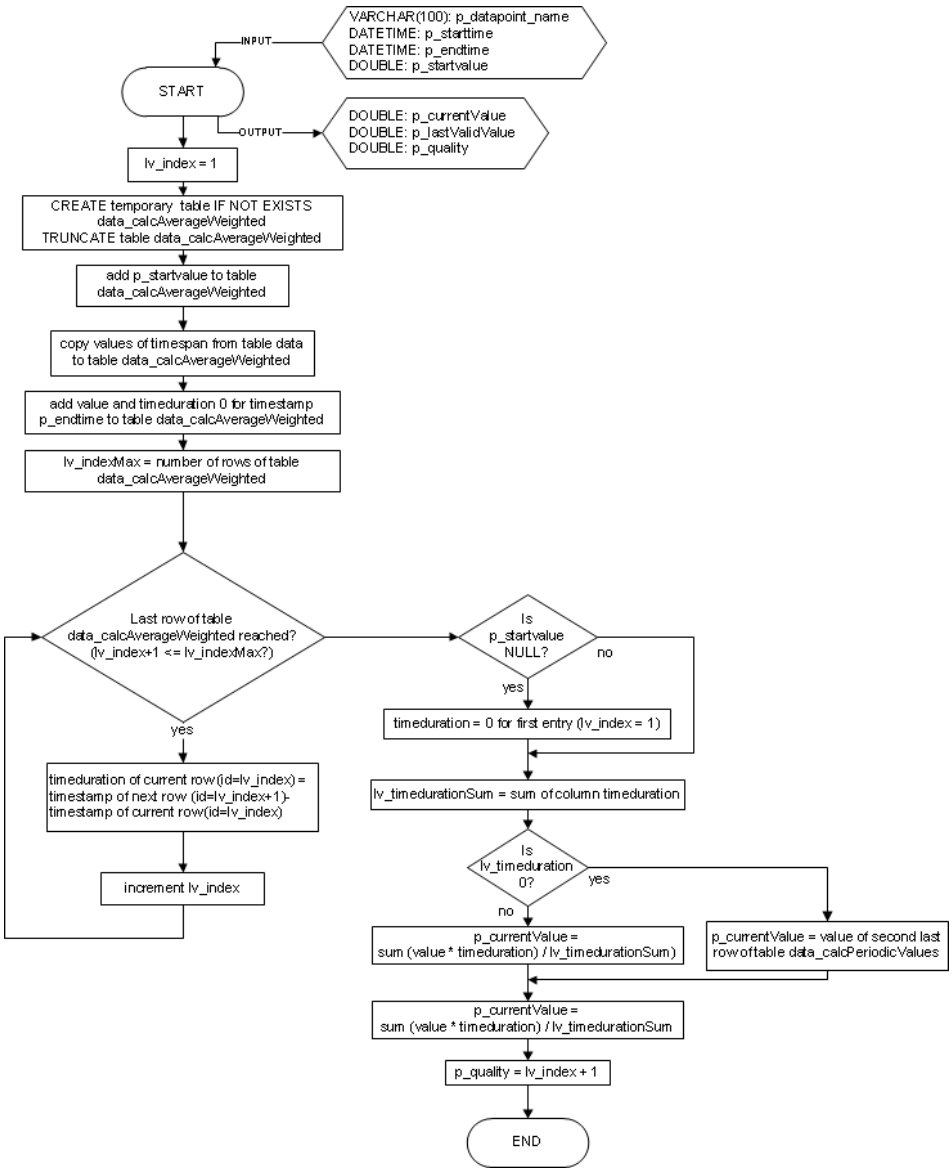


Figure 64. Flow diagram – calcAverageWeighted()

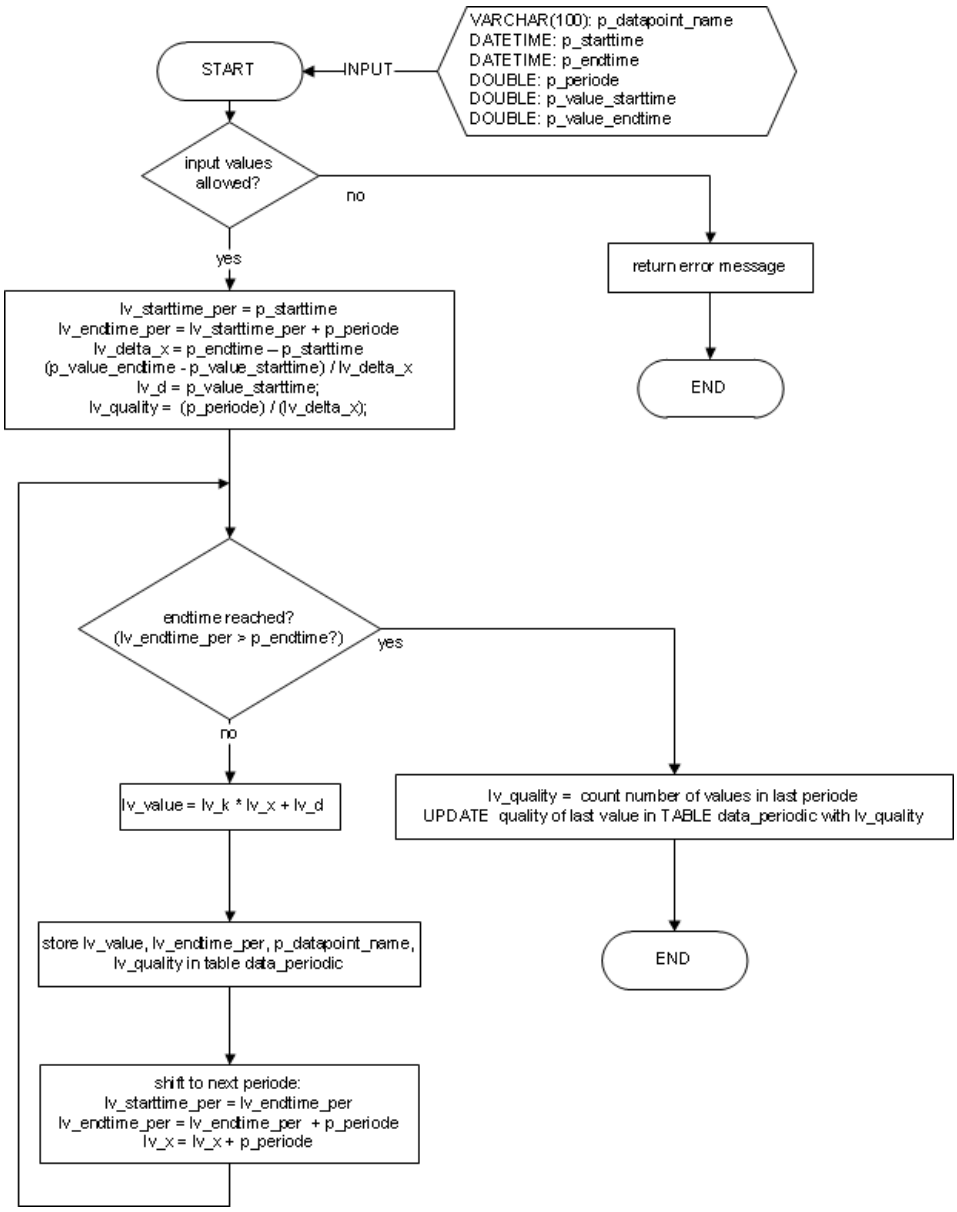


Figure 65. Flow diagram – interpolateValuesLinear()

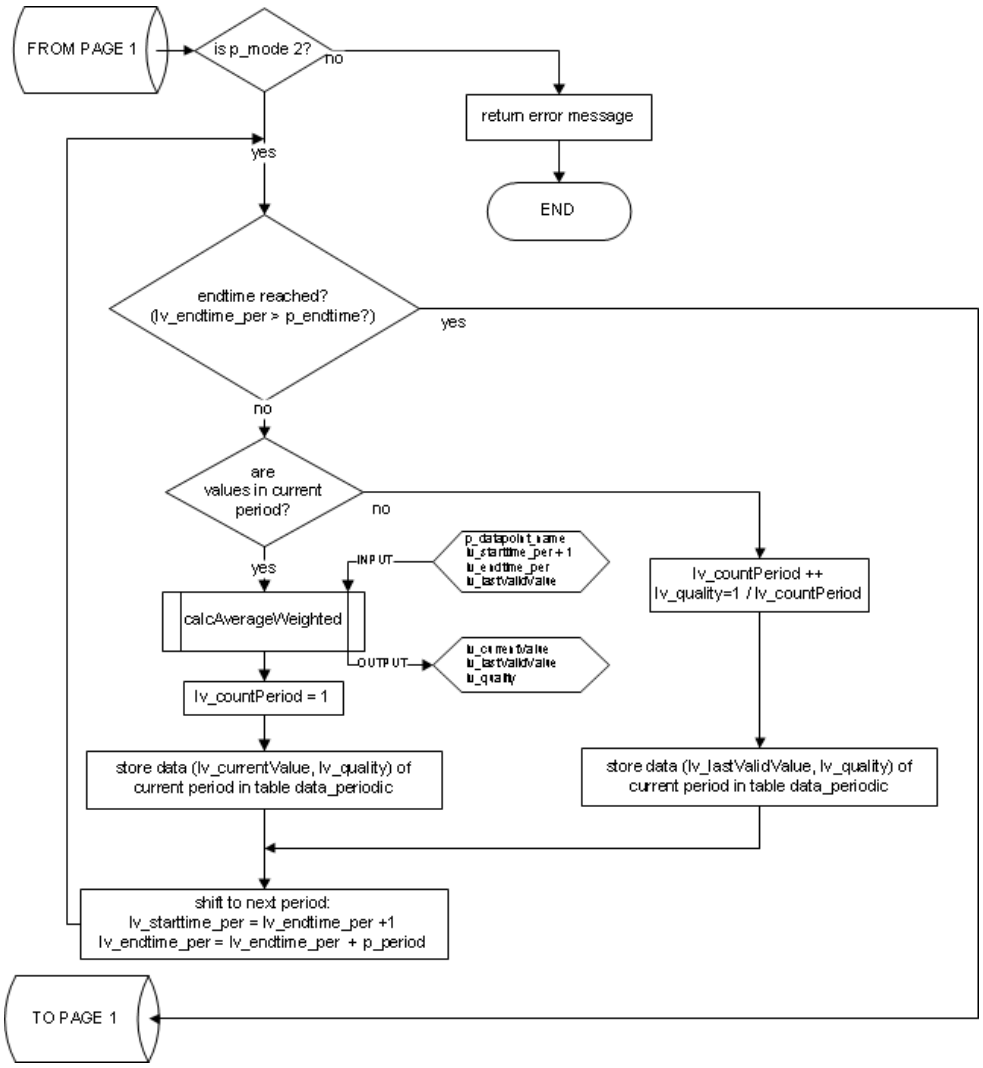


Figure 66. Flow diagram – getValuesPeriodicAnalog()

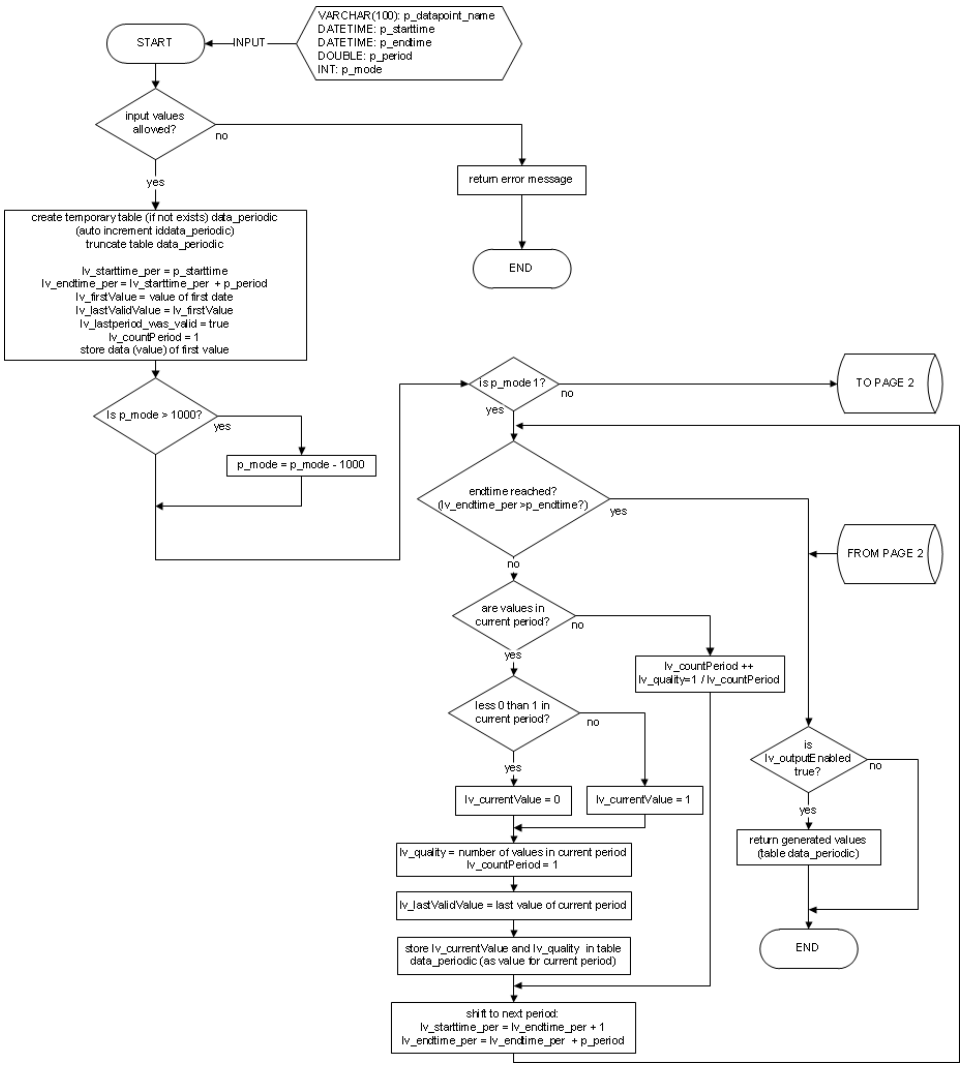


Figure 67. Flow diagram – getValuesPeriodicBinary()

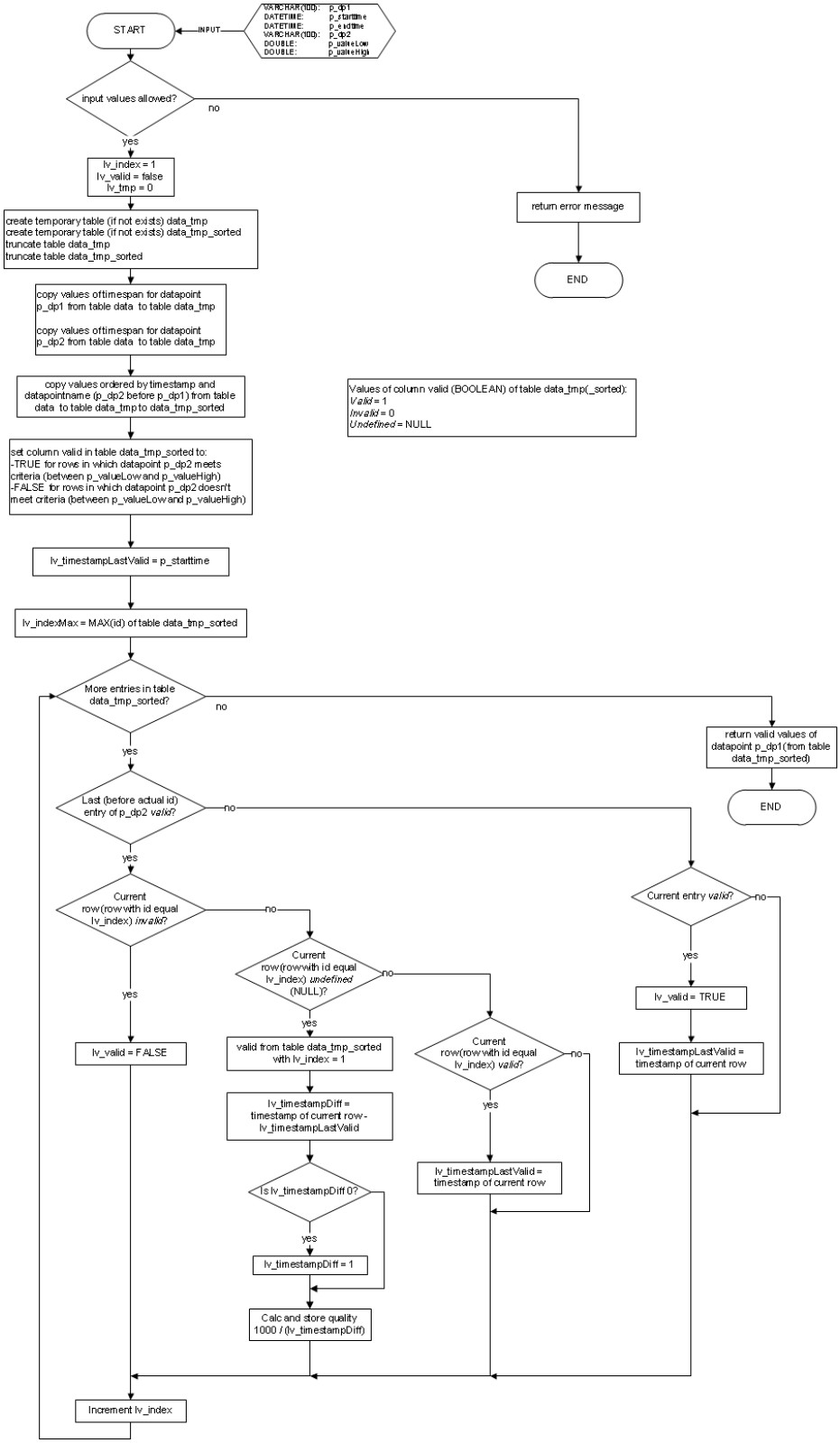


Figure 68. Flow diagram – getValuesWhereDpBetween()

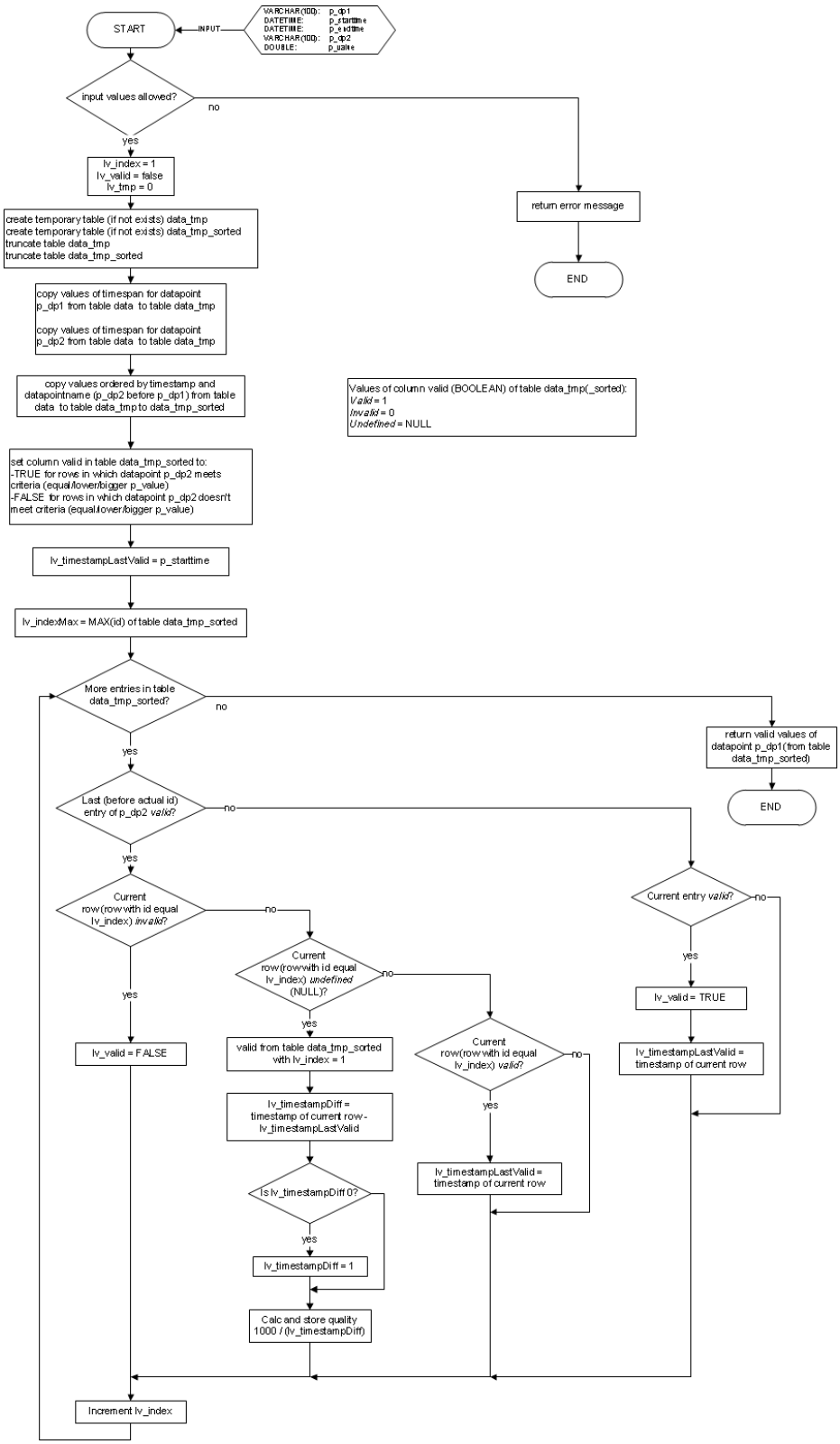


Figure 69. Flow diagram – getValuesWhereDpXXX()

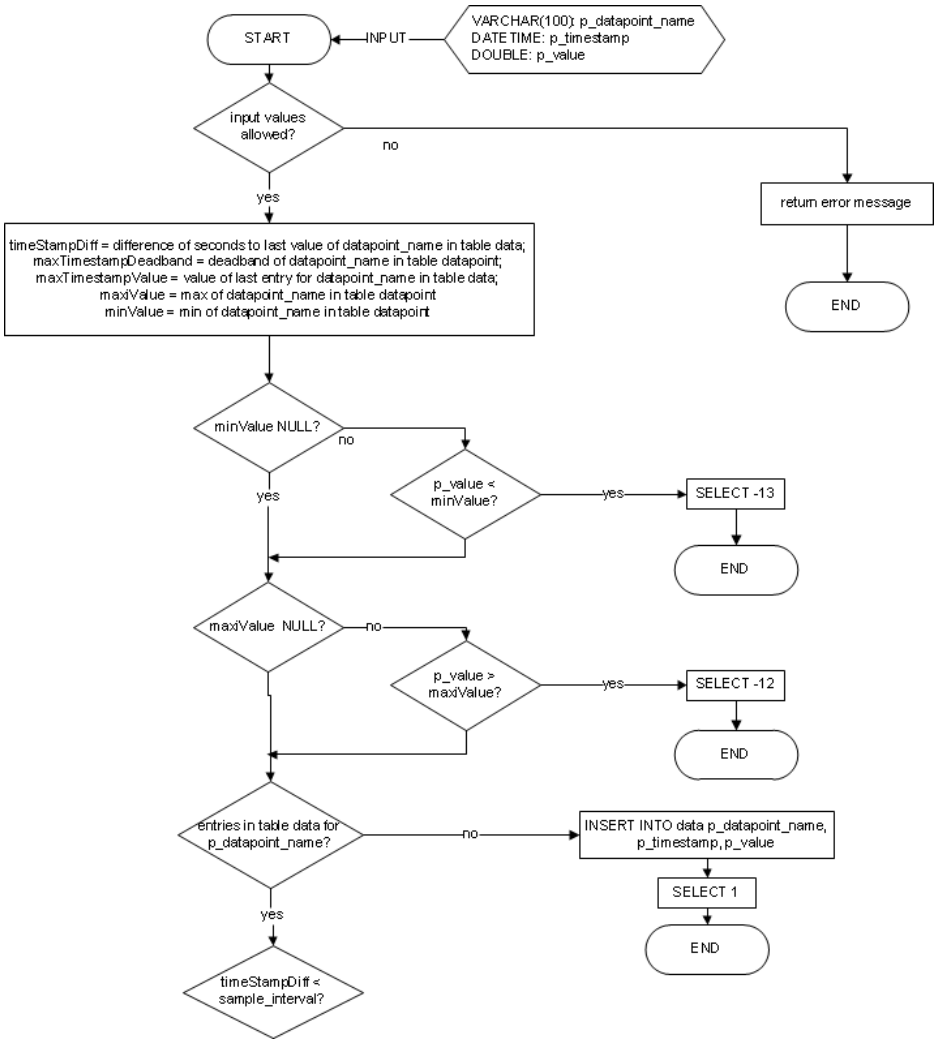


Figure 70. Flow diagram - addData()

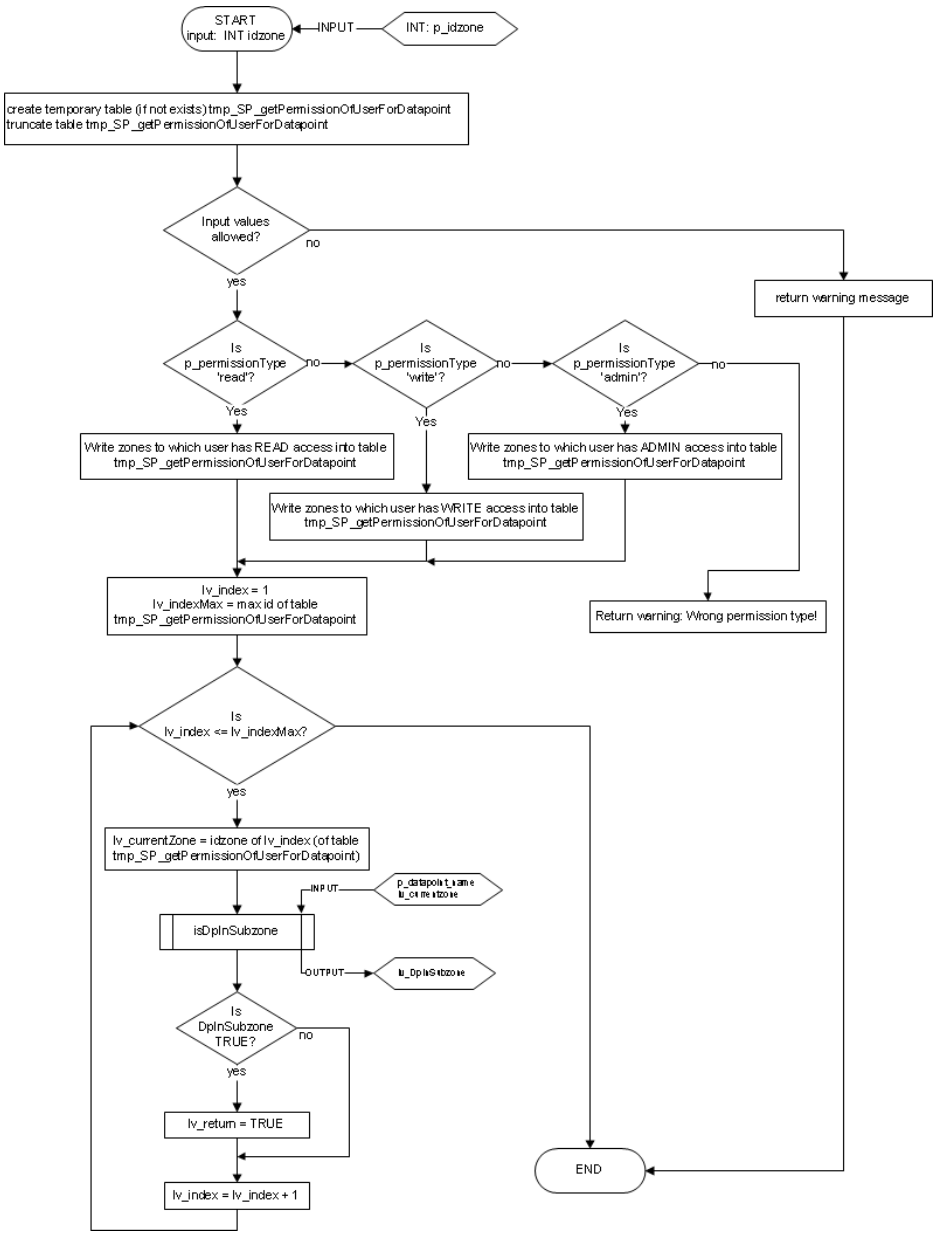


Figure 71. Flow diagram – getPermissionOfUserForDatapoint()

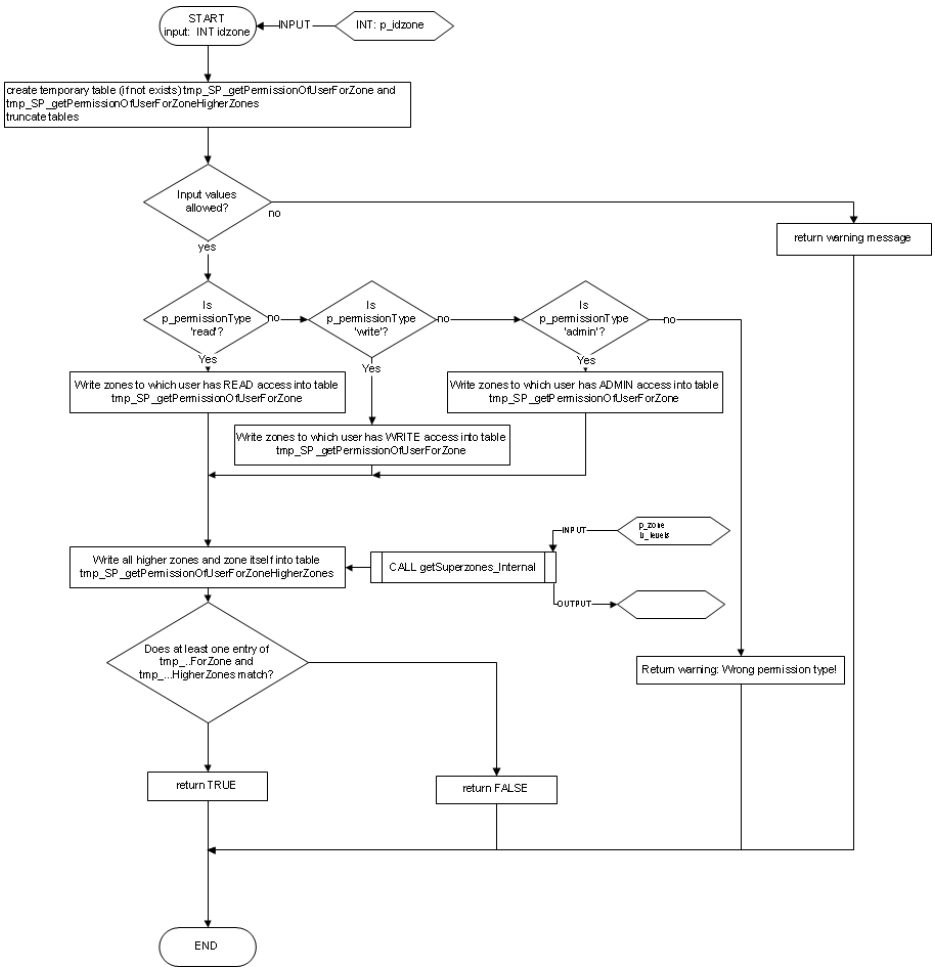


Figure 72. Flow diagram – getPermissionOfUserForZone()

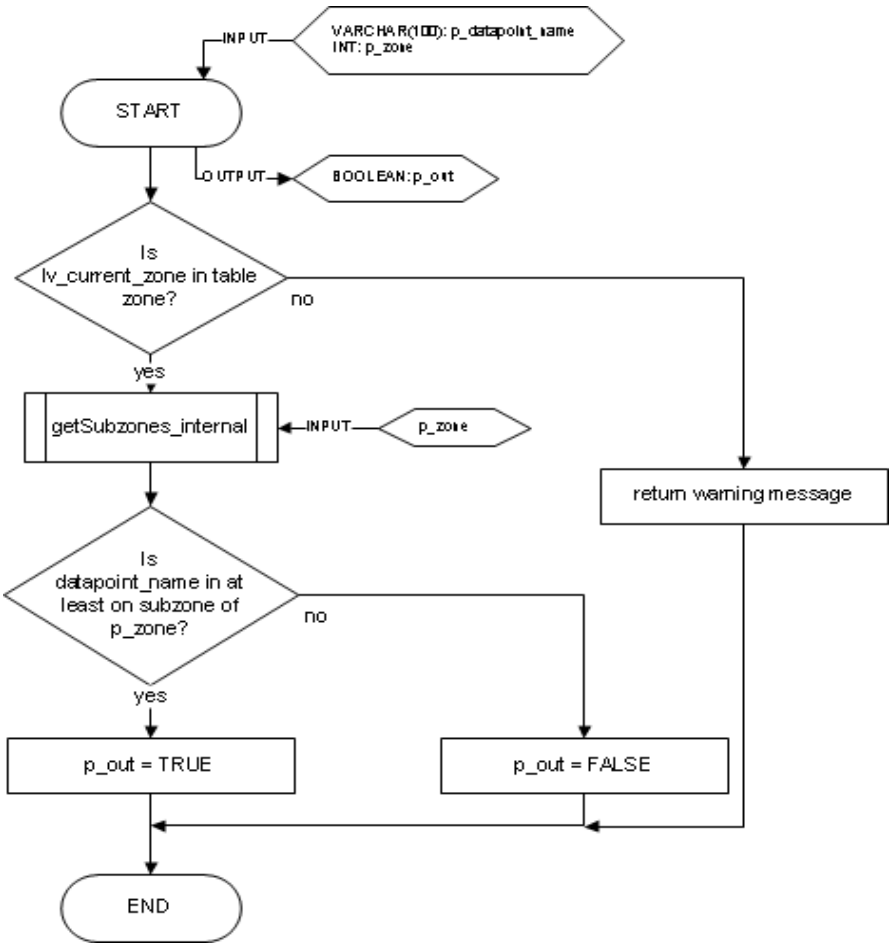


Figure 73. Flow diagram – isDpInSubzone()

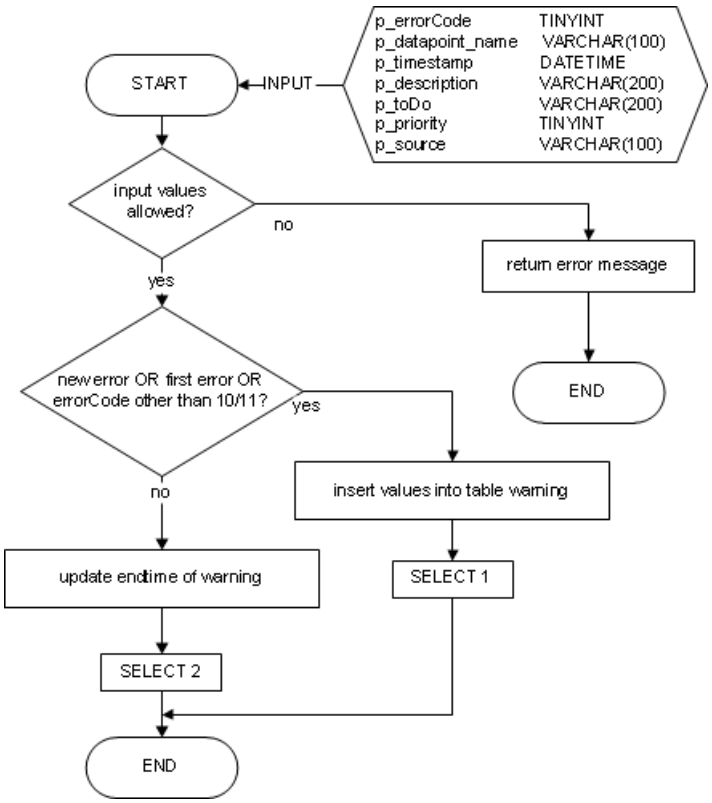


Figure 74. Flow diagram – addWarning()

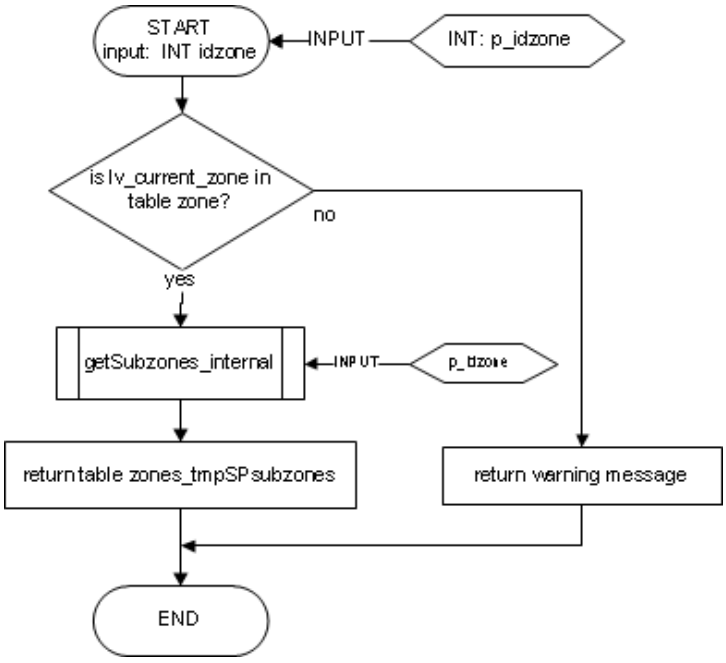


Figure 75. Flow diagram – getSubzones()

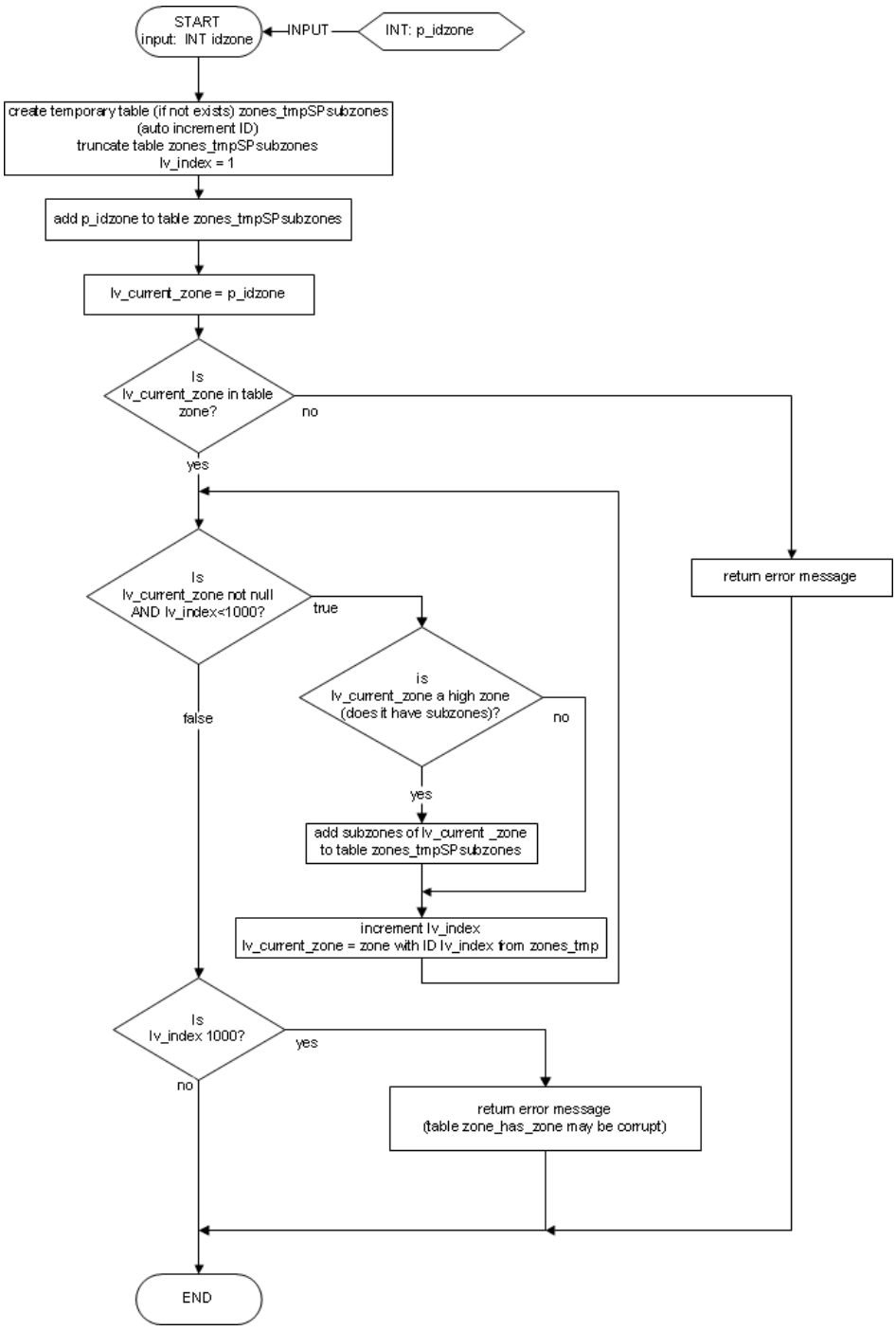


Figure 76. Flow diagram – getSubzones_internal()

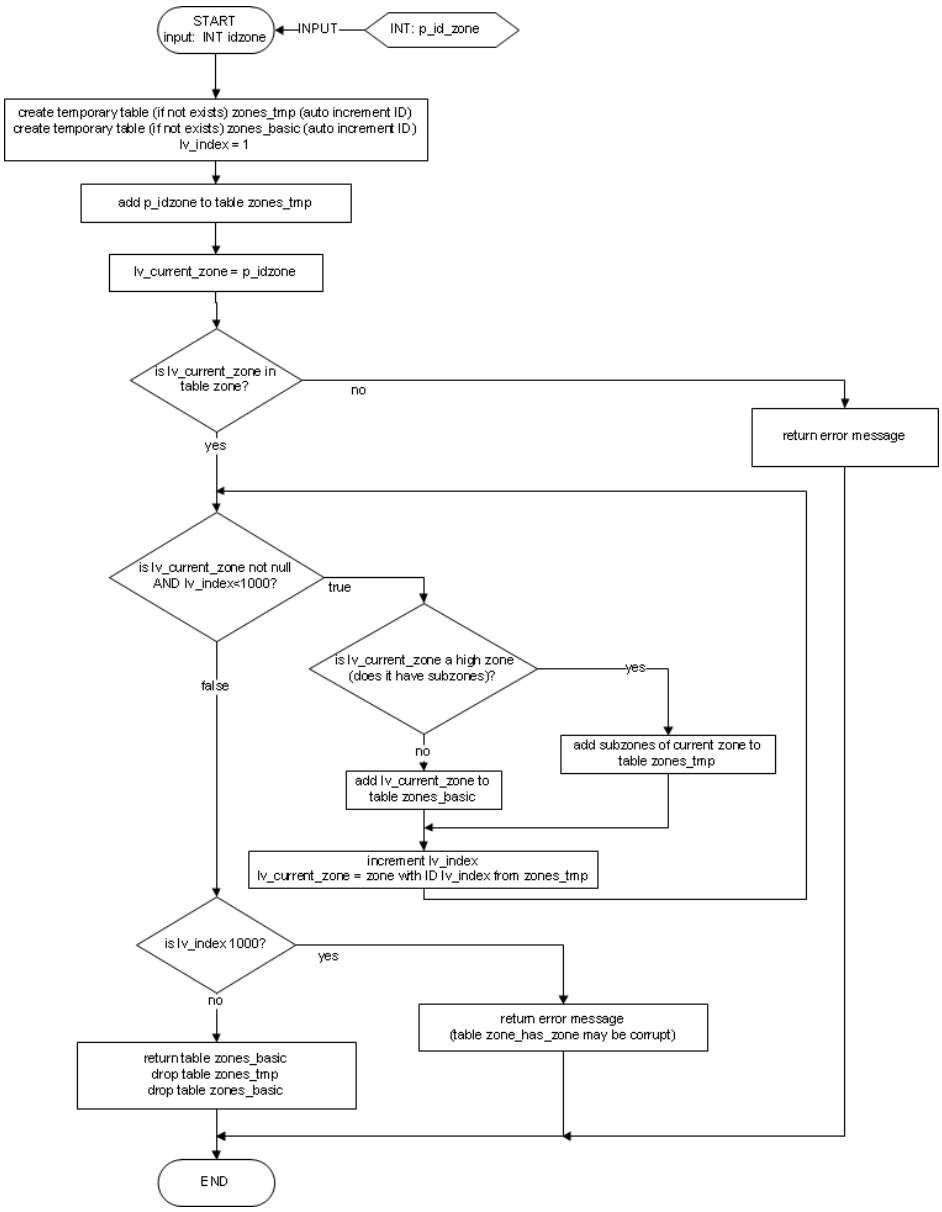


Figure 77. Flow diagram – getBasiczones()

7.2. Virtual datapoint example

The following Java code shows the implementation of the virtual datapoint *RadiatorHeatPower* within the data-abstraction layer. The datapoint calculates the heat power of a radiator based on the mean temperature and the standard heat output of the radiator and the room temperature of the respective zone.

```
package bpi.most.server.model.dpvirtual;

import java.util.Date;
import bpi.most.server.model.Datapoint;
import bpi.most.server.model.DpController;
import bpi.most.server.model.DpVirtualFactory;
import bpi.most.shared.DpDataDTO;
import bpi.most.shared.DpDatasetDTO;

/**
 * ID = "RadiatorHeatPower" RadiatorHeatPower calculates the heat power of a
 * radiator based on the
 * - A) mean radiator temperature
 * - B) the standard heat output of the radiator (based on the respective
 *  radiator size!! --> [W] ), e.g. 1694 W for a common radiator with 1m length
 * - C) (optional) the room
 * temperature of the respective zone (20C is used if not provided).
 *
 * The following custom attributes should be provided:
 * - A) dpMeanTemp - String of dp name
 * - B) standartHeatOutput - Integer, [W]
 * - C) dpRoomTemp - String of
 * Example: dpMeanTemp tem1; standartHeatOutput 1694; dpRoomTemp tem2;
 *
 * @author robert.zach@tuwien.ac.at
 * @parameter virtualDpId = "RadiatorHeatPower"
 * @return A Datapoint Instance or null if the requested type (string id) is
 * not support
 */
public class RadiatorHeatPower extends DpVirtualFactory {

@Override
public Datapoint getVirtualDp(String virtualDpId, String dpName) {
        // if virtualDpId is yours --> return a Datapoint instance
        if (virtualDpId.equals("RadiatorHeatPower")) {
                return new RadiatorHeatPowerImplementation(dpName);
        }
        return null;
}

/*
 * Inner Class with the actual Implementation.
 */
public class RadiatorHeatPowerImplementation extends Datapoint {
        // Custom Attributes
private final String IDdpMeanTemp = "dpMeanTemp";
private final String IDstandartHeatOutput = "standartHeatOutput";
private final String IDdpRoomTemp = "dpRoomTemp";
DpController dpct = null;
private DpDatasetDTO defaultRoomTemp;

public RadiatorHeatPowerImplementation(String dpName) {
        super(dpName);
        // get required DPs
        dpct = DpController.getInstance();
        // create default room temp Dataset - set to 20C
```

```
        defaultRoomTemp = new DpDatasetDTO();
        defaultRoomTemp.add(new DpDataDTO(new Date(), 20.0));
}

/*
 * Overwrite getValues, getValuesPeriodic, etc.
 */

@Override
public int addData(DpDataDTO measurement) {
        // not supported
        return 0;
}

@Override
public int delData() {
        // not supported
        return 0;
}

@Override
public int delData(Date starttime, Date endtime) {
        // not supported
        return 0;
}

@Override
public DpDataDTO getData() {
        // get required DPs
        Datapoint dpMeanTemp = dpct.getDatapoint(getCustomAttr(IDdpMeanTemp));
        // dpRoomTemp is null if not defined
        Datapoint dpRoomTemp = dpct.getDatapoint(getCustomAttr(IDdpRoomTemp));

        // put last value in Dataset
        DpDatasetDTO meanRadiatorTemp = new DpDatasetDTO();
        meanRadiatorTemp.add(dpMeanTemp.getData());

        if (dpRoomTemp == null) {
                return calculateHeatPower(meanRadiatorTemp, defaultRoomTemp)
                                .get(0);
        } else {
                DpDatasetDTO roomTemp = new DpDatasetDTO();
                roomTemp.add(dpRoomTemp.getData());
                return calculateHeatPower(meanRadiatorTemp, roomTemp).get(0);
        }
}

@Override
public DpDatasetDTO getData(Date starttime, Date endtime) {
        // get required DPs
        Datapoint dpMeanTemp = dpct.getDatapoint(getCustomAttr(IDdpMeanTemp));
        // dpRoomTemp is null if not defined
        Datapoint dpRoomTemp = dpct.getDatapoint(getCustomAttr(IDdpRoomTemp));

        if (dpRoomTemp == null) {
                return calculateHeatPower(
                                dpMeanTemp.getData(starttime, endtime),
                                defaultRoomTemp);
        } else {
                return calculateHeatPower(
                                dpMeanTemp.getData(starttime, endtime),
                                dpRoomTemp.getData(starttime, endtime));
        }
}

@Override
public DpDatasetDTO getDataPeriodic(Date starttime, Date endtime,
                Float period, int mode) {
        // get required DPs
        Datapoint dpMeanTemp = dpct.getDatapoint(getCustomAttr(IDdpMeanTemp));
        // dpRoomTemp is null if not defined
```

```
        Datapoint dpRoomTemp = dpct.getDatapoint(getCustomAttr(IDdpRoomTemp));

        if (dpRoomTemp == null) {
          return calculateHeatPower(dpMeanTemp.getDataPeriodic(starttime,
                        endtime, period, mode), defaultRoomTemp);
        } else {
          return calculateHeatPower(dpMeanTemp.getDataPeriodic(starttime,
                        endtime, period, mode), dpRoomTemp.getDataPeriodic(
                        starttime, endtime, period, mode));
        }
}

/**
 * Actual calculation of the heat power Includes smaller quality value
 * of meanRadiatorTemp and roomTemp in returned Dataset
 * @return A Dataset of the calculated heat power
 */
public DpDatasetDTO calculateHeatPower(DpDatasetDTO meanRadiatorTemp,
                DpDatasetDTO roomTemp) {
        DpDatasetDTO result = new DpDatasetDTO();
        DpDataDTO matchingRoomTemp;
        // set dp name in Resultset
        result.setDatapointName(getDatapointName());
        int standartHeatOutput = Integer
                        .valueOf(getCustomAttr(IDstandartHeatOutput));

        // return empty Dataset if not all arguments are valid
        if (meanRadiatorTemp.isEmpty() || roomTemp.isEmpty()) {
                return result;
        }

        // calculate power value for each radiator meanTemp measurement
        for (DpDataDTO meanTempData : meanRadiatorTemp) {
                Double power = null;
                Double diffTemp = null;
                // ### get matching room temperature
                // use roomTemp measurement before or equal to the respective
                // meanTemp measurement timestamp
                matchingRoomTemp = roomTemp.getDataBeforeOrEqual(meanTempData
                                .getTimestamp());
                //if no data before or equal use after, only required for
                //special case where you have no measurements in the beginning
                if (matchingRoomTemp == null) {
                        matchingRoomTemp = roomTemp.getDataAfter(meanTempData
                                        .getTimestamp());
                }

                // ### calc diff. temp.
                diffTemp = meanTempData.getValue()
                                - matchingRoomTemp.getValue();
                // negative values caused by measurement faults are set to 0
                if (diffTemp < 0.0) {
                        diffTemp = 0.0;
                }
                // calc heat power - see documentation
                power = 0.0062 * java.lang.Math.pow(diffTemp, 1.2998)
                                * standartHeatOutput;
                // set smaller quality value
                if (meanTempData.getQuality() < matchingRoomTemp.getQuality())
                        result.add(new DpDataDTO(meanTempData.getTimestamp(),
                                power, meanTempData.getQuality()));
                else
                        result.add(new DpDataDTO(meanTempData.getTimestamp(),
                                power, matchingRoomTemp.getQuality()));
        }
        return result;
}


/*
 * @see
```

```
 * bpi.most.server.model.Datapoint#getNumberOfValues(java.util.Date,
 * java.util.Date)
 */
        public int getNumberOfValues(Date starttime, Date endtime) {
                return dpct.getDatapoint(getCustomAttr(IDdpMeanTemp))
                                .getNumberOfValues(starttime, endtime);
        }
    }
}
```

7.3. MATLAB-Framework

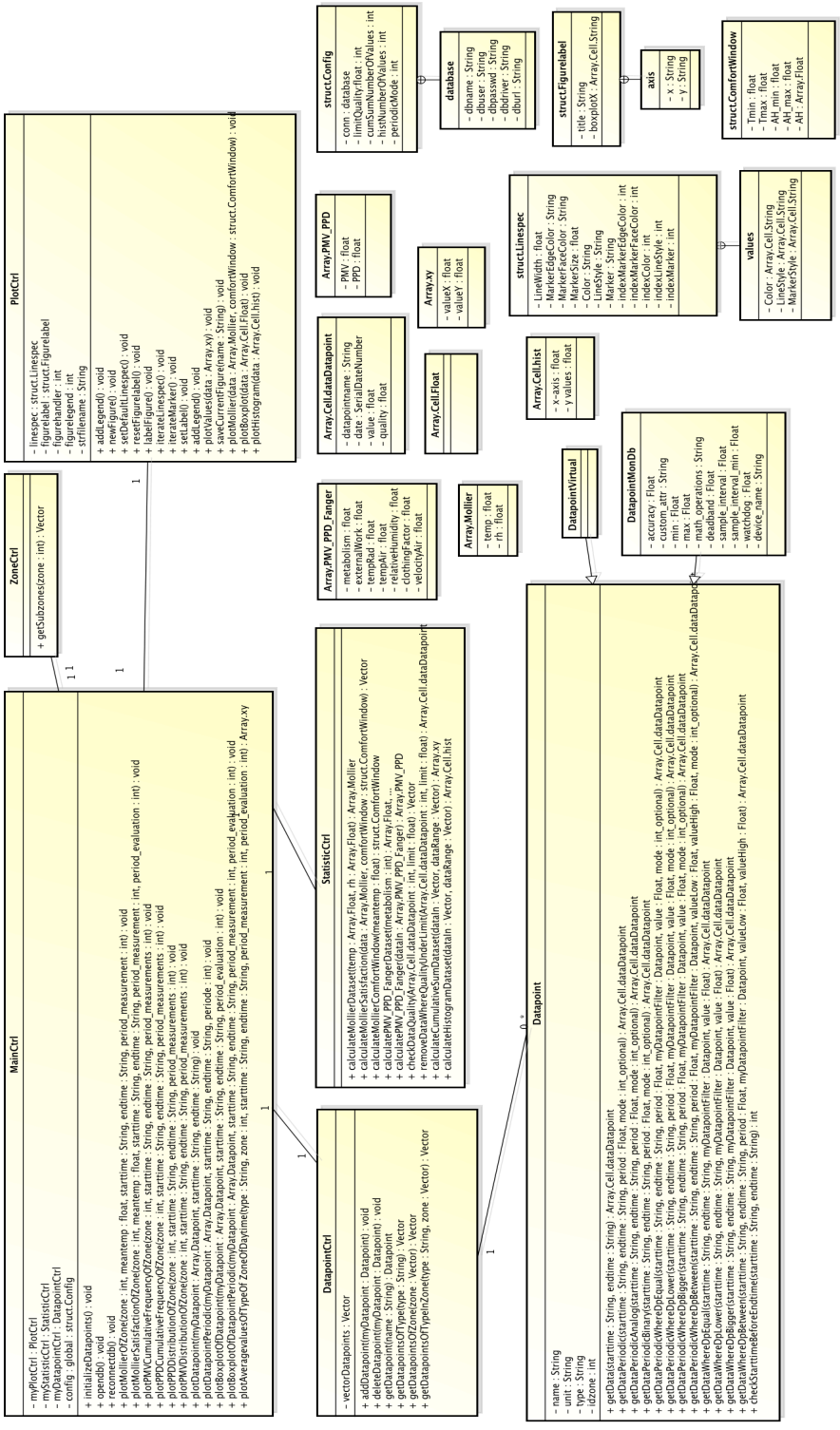


Figure 78. Complete class diagram of the MATLAB-Framework

CV - Robert Zach

Professional Background

2009 -	Univ.Ass., Department of Building Physics and Building Ecology, Vienna University of Technology, Austria
2006 - 2009	Embedded Systems trainer and engineer. So-logic electronic consulting and Xilinx Inc.
2002 - 2006	Technical Support for Silberpfeil architects, SAPI consulting, Phonetastic consulting and Bioservice Zach Gmbh

Educational Background

2009	Mag.rer.soc., Vienna University of Technology, Austria (with distinction)
2007	Dipl.-Ing.(FH), University of Applied Science, Austria (with distinction)
2006	(Study abroad) University of California, UCLA, Los Angeles, USA
2006	Sun certified Java Programmer
2004	HTBLA Vienna 10, College for Electronics and IT (with distinction)
2003	Cisco CCNA certified
2002	HTBLA Karlstein/Thaya, Commercial School for Microelectronics

Research Areas

Building physics, informatics, building automation, embedded systems