Recent advances in the development of MOST: an open-source, vendor and technology independent toolkit for building monitoring, data preprocessing and visualization

Stefan Glawischnig, Regina Appel, Robert Zach and Ardeshir Mahdavi

Department of Building Physics and Building Ecology, Vienna University of Technology, Austria

Email: stefan.glawischnig@tuwien.ac.at

Abstract:
The present contribution describes approaches for visualizing building related data (temperature, energy use, etc.). A web based visualization framework is presented and a number of use cases are demonstrated (i.e. three-dimensional building browsing). Usability is optimized for diverse screen sizes, input methods (i.e. touch screen), and application logics. Finally, guidelines for further user interface development are presented.

Keywords:
Building management system, monitoring, building data visualization, toolkit, MOST

1. Introduction

The present research is based on a vendor and technology independent toolkit for building monitoring, data processing and visualization. On the one hand this Monitoring System Toolkit (MOST) focuses on the maintenance of a comprehensive data collection, real time data access and the integration of data preprocessing and aggregation techniques. On the other hand diverse software interfaces enable multiple applications to process desired data streams. This paper focuses on one possible processing application, a web-based user interface, which visualizes building information and enables human interaction. The web offers the opportunity to access multiple, spatially distributed physical and virtual data sources in a standardized way. The presented visualization platform

- supports multiple interaction devices (mouse/keyboard, touch, gestures),
- provides reusable user interface elements (charts, menus, etc.),
- separates diverse use cases in different modules and includes a number of prototypical implementations.

Previous work covers real-time building data collection (Zach et al. 2012a), preprocessing and data aggregation (Zach et al. 2012b), and the investigation of possible use cases (Chien et al. 2011).

1.1. Approach

The aim of this work is to build a web based application that provides access to the proposed building monitoring toolkit. Using a prototypical implementation, usability tests are examined. Figure **1** shows the three-layer software architecture of the proposed web interface. On the top level, the user primary interacts with the graphical interface via drag and drop actions. The interface increases intuitive operation by highlighting areas where operations can

be performed on dragged objects. Underneath, application logic is separated into modules that can reuse graphical elements of a generic library (*MenuWidget*, *DatapointWidget*, etc.). Operations are supported by control instances that connect graphical elements to certain logic (*ModuleCtrl*, *DndCtrl*, etc.). All communication to the monitoring server is covered within the abstraction layer.

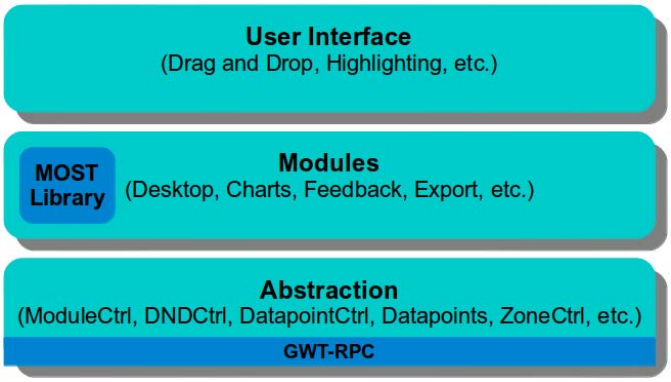

Figure 1. MOST client side framework layers

2. Visualization framework

To enable platform independent user interface development, web based technologies can be used. Furthermore, web frameworks (JSF, GWT, etc.) simplify development by abstracting specific web technologies. 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. The overall framework focuses on the following design principles (Dix 2004):

- Useful (appropriate functionality)
- Usable (user is able to perform tasks)
- Used (attractive and available).

2.1. Modules

Independent modules can be used to implement different use cases within the visualization framework. All modules are controlled by the class *ModuleController* as shown in Figure **2**. Each module must implement the Java interface *ModuleInterface* and the abstract class *ModuleWidget*. The *ModuleInterface* defines certain meta-data, which provide module specific information (module name, URL, etc.). The graphic representation derives from the abstract class *ModuleWidget*. Each module is instantiated once by the *ModuleRegistrator* class at runtime. During this process, authentication and security procedures are executed.

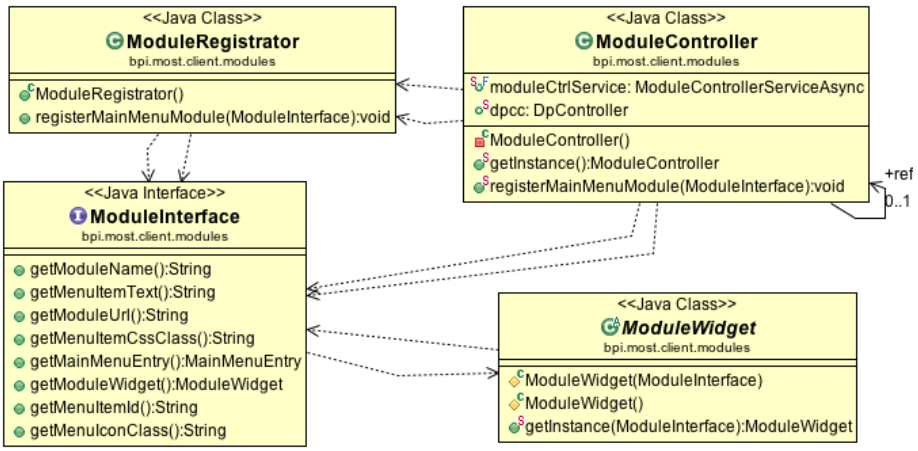

Figure 2. Module registration process

2.2. User interface library

The proposed user interface library provides various components for simplifying module development and building platform and input optimized user interfaces. Nielsen and Molich (1990) state that human-computer interaction is error prone and that the user should be supported when making a decision. Including these premises, the application restricts and guides the user input with two mechanisms, drag and drop and highlighting. User input applies drag and drop as the primary interaction method. Interaction processes become more dynamic and intuitive. By visually highlighting droppable areas the user's choices are limited to valid options. By eliminating erroneous interaction possibilities the application design is improved according to Nielsen's ten usability heuristics (Nielsen 1994).

The proposed library consists of graphical components and logical elements (see Figure **3**). Components are classes, which provide a graphical representation (i.e. *xxxWidgets*). Logical elements are classes, which include no graphical part, but diverse application functionality (i.e. *DragInterface*). The library is based on native GWT features and can be integrated in any GWT project.

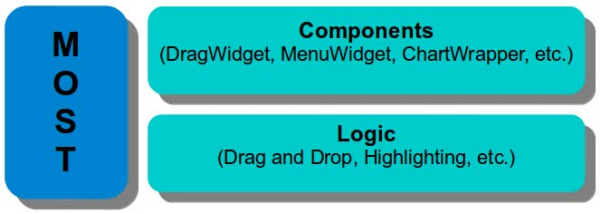

Figure 3. User interface library

2.3. Drag and drop

All available widgets natively support the introduced drag and drop mechanisms (see Figure **4**). The class *DNDController* implements highlighting by adding and removing Cascading Style Sheets (CSS) information from the Document Object Model (DOM). Each *DragWidget* contains information about compatible *DropWidget*. The *DNDController* constantly listens for drag and drop operations and highlights respective *DropWidget* on demand. Each GWT element can be used with the proposed drag and drop features by extending the class *DragWidget*. Several reusable user interface objects are provided within the library (menus, windows, etc.) to unify the user interface design.

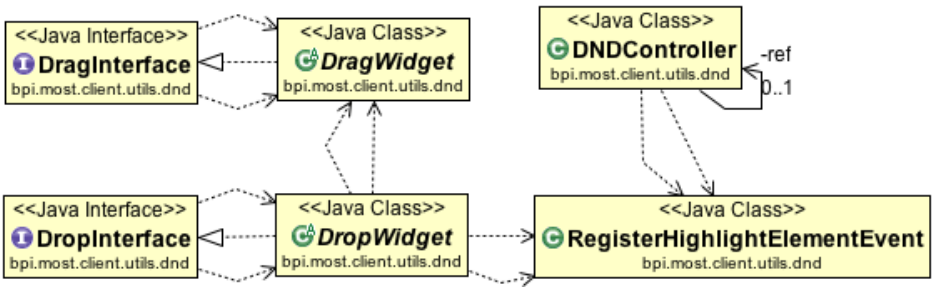

Figure 4. Drag and drop logic

Following these design principles, the visual implementation is separated from the application logic. Figure **5** shows the proposed drag and drop/highlighting feature in a showcase application.

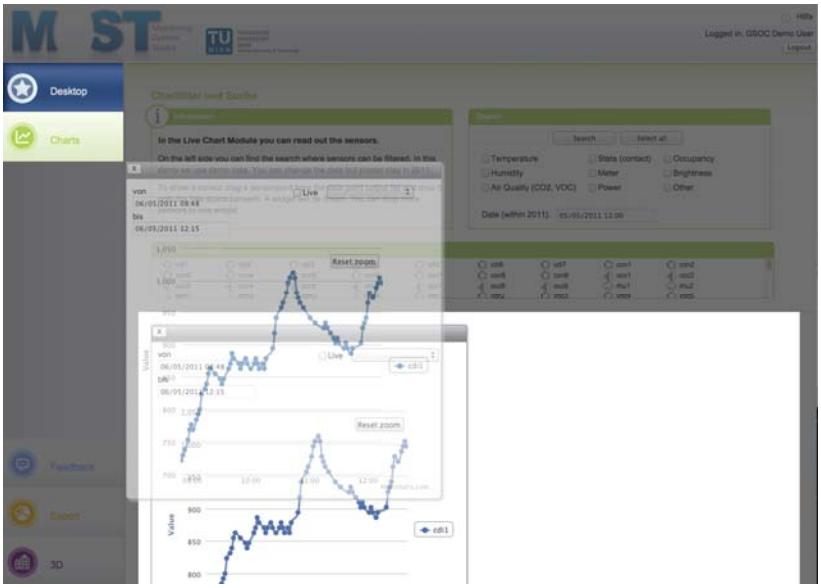

Figure 5. Drag and drop/highlighting procedure

3. Application Modules

To show the potential of the visualization framework, some modules covering various use cases were implemented. For example, the module "3D-Viewer" is provided to map available information points (datapoints) within a three-dimensional model of a building. Furthermore, the "Chart Module" can be used to plot a trend chart of a desired datapoint. A "Desktop" module is available to accomplish different information sources on a central page (similar to the Microsoft Windows Desktop). Diverse objects can be exchanged between modules using drag and drop operation. When dropped on predefined areas, the corresponding module's specific code processes the incoming information. Figure **6** shows an exemplary use case. First a desired datapoint is found within the 3D-Viewer. Then, a trend chart is generated by moving the respective datapoint to the Chart module (a). Finally, the resulting chart is moved to the Desktop module (b).

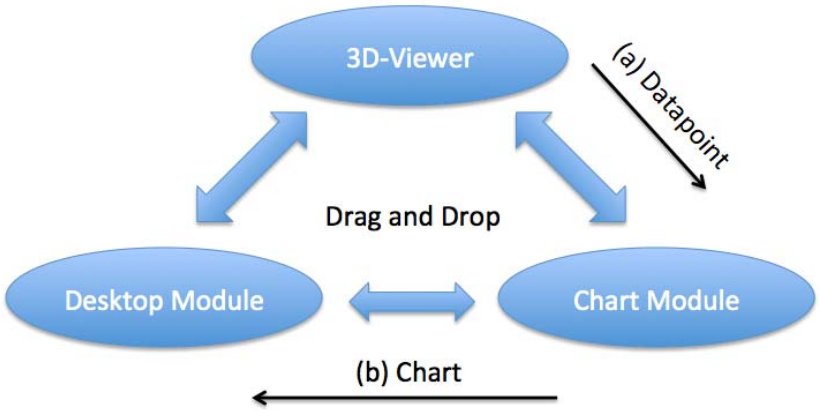

Figure 6. Inter-module communication process

3.1. 3D-Viewer

The module "3D-Viewer" 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. It 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. It showed appropriate performance characteristics. Therefore, BIMsurfer 2012 was chosen to be enriched with building monitoring related features. 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
B. Show the user the location of a particular datapoint
C. Visualize critical information of datapoints (i.e. errors)

To lead the user to the desired datapoints in an intuitive way, additional building model browsing features were implemented. For example, building stories can be exposed using a slider bar as shown in Figure **7**. If no building story is marked, all levels are shifted. If a single floor is selected, only this one is exposed. Furthermore, the transparency of the building walls can be controlled with an additional slider bar. The building model is described within a hierarchically structured JSON file. Each building element (door, wall, etc.) is represented as a SceneJS node. Exposing works by translating a node and the children of the respective node along the y-axis by a specific value. Inserting and removing transparent nodes into the building tree changes the transparency of all child nodes. The developed BIMSurfer fork is integrated into the proposed framework by injecting the JavaScript code into GWT code at runtime.

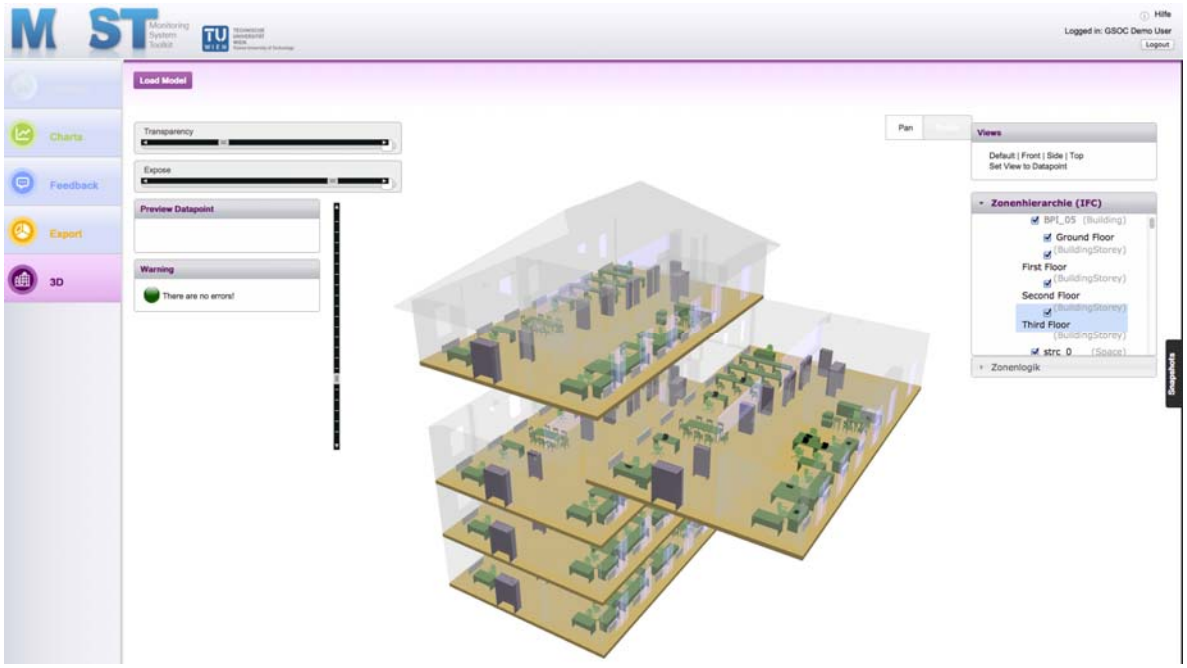

Figure 7. 3D-Viewer module interface

Use case B (show the location of a particular datapoint) is supported by dynamically moving the view to the location of a requested datapoint. The implementation is based on the snapshot feature of BIMsurfer 2012. When dropping a datapoint object on the 3d model, the viewer automatically centers the view on the desired node. Use case C (Visualize critical information of datapoints) is prepared but still under development. It is intended to load warnings of datapoints, within the visualized building model cutout, on demand.

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 CAD application deployed 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.

3.2. 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 *GeneralDragWidget* as shown in Figure **8**. 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 MOST toolkit - are used to calculate a reduced number of values on demand. Before drawing a chart, the amount of measurements in the requested timeframe is analyzed. If a critical amount of measurements is exceeded, data preprocessing algorithm (Zach et al. 2012c) are 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 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. Via the proposed drag and drop approach, new datapoints can be dragged from various source modules (chart, 3D-Viewer, etc.) and dropped on an existing chart. The chart includes the new source automatically. Enabling the live feature triggers the server to push new values to the web interface and to include them in the existing chart.

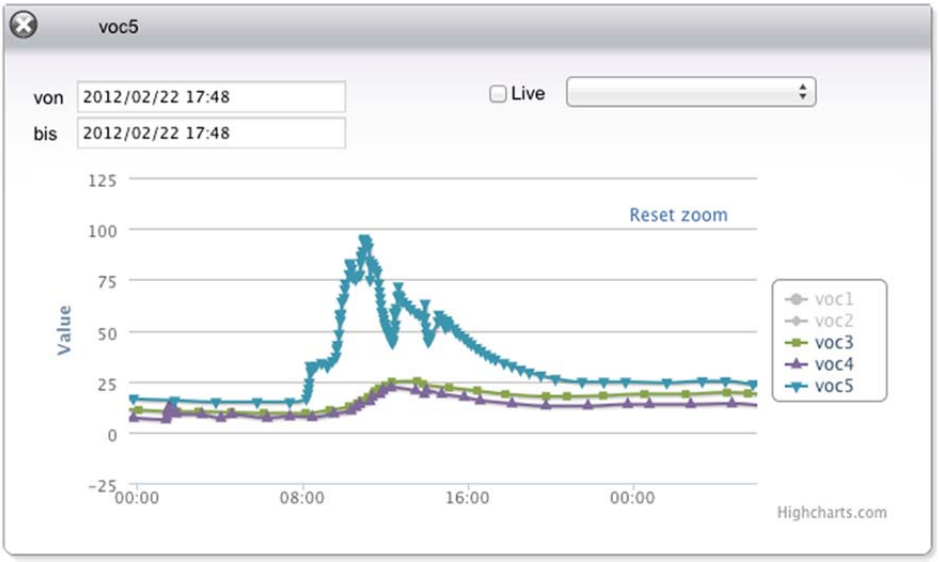

Figure 8. *DragWidget* object containing a chart with multiple datapoints

4. User Interaction

The user interface was optimized for three different types of user input technologies, to support a wide range of devices and to broaden the range of use cases. These user interface types include established methods (mouse/keyboard and touch input) and one upcoming technology, the gesture input (natural interaction). The target output devices are tablets, desktop PCs, and information screens (i.e. touchscreen or beamer).

4.1. Input Interaction

To enable similar interaction processes for all input devices, the following operation types are defined:

- Select/activate
- Drag
- Pan/rotate
- Zoom

Select and drag operations build the basis for all interaction processes. Pan, rotate, and zoom are advanced input methods, which are only required in certain cases (i.e. 3D-Viewer) and are not natively supported by all input devices. Table 1 to 4 show how diverse interaction operations are implemented with different input devices.

Table 1: Interaction type select/activate

Input device	**Interaction**
Mouse/ keyboard	Clicking
Touchscreen	Tapping
Gesture input	Push gesture (the hand is moved in the direction of the sensor like pushing an imaginary button before the user)

Table 2: Interaction type drag

Input device	**Interaction**
Mouse/keyboard	Clicking plus dragging the selected widget to the target point and releasing it
Touchscreen	Tapping plus dragging the selected widget to the target point and releasing it
Gesture input	Push gesture plus dragging the selected widget to the target point and releasing it

Table 3: Interaction type pan/rotate

Input device	**Interaction**
Mouse/keyboard	Clicking on the building model plus holding down the mouse button and moving the cursor
Touchscreen	Tapping on the building model plus holding down the tip and moving the cursor
Gesture input	Push gesture on the building model plus keeping the posture and moving the hand

Table 4: Interaction zoom

Input device	**Interaction**
Mouse/keyboard	Using the scroll wheel of the mouse or two fingers on a track pad
Touchscreen	Single touch: Using a zoom slider. Multi touch: Putting two fingers on the screen and moving them apart/closer together.
Gesture input	(1) Changing the zoom slider in the user interface (2) Moving the body closer/farther from device to zoom in or out (3) Move both hands apart/closer

Pan and rotate functionalities are based on the same interaction procedure. A switch to distinguish between the two navigation modes is implemented within the 3D-Viewer module. Input device based differentiation between pan/rotate is part of future research.

4.2. Usability Guidelines

We intend to conduct a usability study of the user interface environment in the near future. One focus of this study is to generate reusable guidelines for the implementation of touch and/or gestural interaction compatible user interfaces. Therefore, different user groups are tested using the proposed visualization framework. The analysis process is based on the usability engineering lifecycle defined by Mayhew 1999. It includes various steps, which are:

- Requirements analysis (includes persons, task analysis, general design principles and usability goals),
- Design/testing/development (includes wireframes, storyboarding, heuristic evaluation, screen design standards, detailed user interface) and
- Installation setup.

Test users conduct predefined tasks in the proposed application environment. The process is monitored and usability bottlenecks are analysed. Qualitative usability goals for all input devices are

- untrained users are able to select datapoints from the 3D-Viewer,
- users are able to create a chart plot of a datapoint and
- users are able to drag and drop a widget from a module to another one.

By analysing the usability bottlenecks, the user interface can be optimized based on the three points of Dix 2004. It should be useful (functionally), usable (it is easy to do things), and used (attractive and available).

5. Conclusion and future outlook

The proposed framework offers a bundle of components that simplifies the development of dedicated, platform optimized user interfaces. The concept focuses on reusability and usability. Therefore multiple screen sizes and input methods were tested. By conducting a usability study, guidelines will be developed to provide a basis for future user interface design. Core questions are:

- How can interaction operations be ported to the proposed input devices?
- Which input operations are best suited for a specific input device?
- Which constraints (i.e. hardware, driver) must be considered?

6. Acknowledgement

The research presented in this paper is supported by funds from the program “Neue Energien 2020” within the “Klima- und Energiefonds” (no: 834517). 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 re-search objectives. Information on further developments regarding the proposed monitoring and visualization toolkit is available at http://most.bpi.tuwien.ac.at.

References:

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*

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

Dix A. 2004, *Human Computer Interaction*, Pearson, Harlow, England.

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

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/*

Mayhew D. 1999, *The Usability Engineering Lifecycle*, Morgan Kaufmann Publishers, San Francisco, Calif.

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

Nielsen J. and Molich, R. 1990, *Heuristic evaluation of user interfaces*, Proc. ACM CHI'90 Conf. (Seattle, WA, 1-5 April), 249-256.

Nielsen J. 1994, *Heuristic evaluation*, In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods. John Wiley & Sons, New York, NY.

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*

Zach R., Glawischnig S., Hönisch M., Appel R., Mahdavi A. 2012a, *MOST: An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization*, 25 – 27 July, Reykjavik, Island

Zach R., Schuss M., Bräuer R., Mahdavi A. 2012b, *Improving building monitoring using a data preprocessing storage engine based on MySQL*, 25 – 27 July, Reykjavik, Island

Zach R., Glawischnig S., Appel R., Weber J., Mahdavi A. 2012c, *Building data visualization using the open-source MOST framework and the Google Web Toolkit*, 25 – 27 July, Reykjavik, Island