A workflow for managing building information and performance data using virtual reality: an alternative to BIM for existing buildings?

Adam Rysanek, Clayton Miller, Arno Schlueter
Chair of Architecture and Building Systems
Institute of Technology in Architecture, ETH Zurich, Zürich, Switzerland

Abstract

Building information modelling (BIM) has carved a growing niche in the construction industry for the support of new building projects. The same cannot be said for existing buildings, where the prevalence of uncertain data, and unclear information, has been difficult to reconcile with the unambiguous nature of BIM parameterization. An opportunity to alleviate these challenges may have arrived from the recent boon of virtual reality platforms for navigating physical environments. This paper demonstrates how labeling of equirectangular images with data or text widgets is possible using publicly-available software libraries. A prototype is presented and tested on a building in Singapore.

Introduction

Since the early 2000s, building information modelling (BIM) has carved a growing niche in the construction planning industry for the development and support of new building projects. To the uninitiated, BIM can be described as an interdisciplinary, structured systems design and modelling framework that allows for precise, physics-centric planning of architectural, structural, mechanical, and electrical systems (Miettinen and Paavola, 2014). How this framework is interpreted and used in practice does vary, however, and is attributed to differing technical capabilities of individual commercial tools and user requirements. A detailed review of modern BIM tools and their capabilities for new building design is provided by Azhar (2011).

The focus of this paper is the application of BIM specifically to the *existing* building - a topic that has emerged recently as its own research subdomain. Volk et al. (2014) provide a comprehensive review of recent research literature on the application of BIM to existing buildings. The authors also provide a relevant enumeration of previously-stated functionalities of BIM in regards to the existing building. This publication infers that the most-cited features of BIM for existing buildings, whether they already exist or are being proposed, are:

- Documentation, data management and visualization
- Energy, thermal analysis, and lighting simulation
- Deviation analysis, quality control, and defect detection
- Localization of building components and indoor navigation
- Life-cycle assessment
- Facilities management, operations and maintenance
- Monitoring and performance measurement

As stated in Volk et al. (2014), the prevalence of uncertain data and unclear or hidden semantic information has been difficult to reconcile with the unambiguous nature of BIM parameterization. The creation of a BIM model for an existing building is considered to be an inherently more complex and costly exercise than an application to a new building. For existing buildings, a BIM modeller is likely to have more insufficient or inadequate documentation about the existing built environment, especially for the oldest of buildings. Ilter and Ergen (2015) draw a similar conclusion.

Interesting technology solutions to these challenges have been proposed. For example, Jung et al. (2014) presents a promising, sophisticated method of interpreting scanned, 3D point cloud data of interior spaces in order to construct a digital model that can be interpreted and incorporated into professional BIM software such as Revit. A review of similar approaches developed in prior research is undertaken by Tang et al. (2010).

In this paper, we propose that there may already exist a completely alternative suite of software tools and libraries that can formulate user-building interfaces with many of the features of BIMs that are stated by Volk et al. to be desirable for existing buildings. Our proposal relies on a complete replacement of the typically bottom-up, closed-source engineering software architecture with a long-used web-based software development framework: the mashup.

Proposal and scope of paper

This paper will demonstrate how recently developed open-source JavaScript (JS) frameworks and application programming interfaces (APIs) for virtual reality applications have made it possible to create, with relative ease, a software 'mashup' providing similar functionality to present-day BIMs - at least for existing buildings. In the following sections, we describe the structure of an example tool, the APIs utilized, and explore capabilities of the tool through application of a case study. It will also be shown that one of the key enablers of this work, and the presented case study, has been the implementation of a building management system featuring remote accessibility of real-time data from the web. Overall, the authors aim for this paper to be an introductory effort on the development of VR-based applications in the building sciences using the latest generation of software and hardware tools. Hence, more emphasis is placed on describing the basis for the developed workflow, and its programming components, rather than proving the capabilities of the developed example tool against one or more existing BIMs. Such an undertaking must form part of a necessary future work.

Background

The mashup: a powerful web-based strategy for next generation building performance modelling tools

Mashups are web-based software tools that rely primarily on pre-existing open-source software code to perform the majority of their intended functions or tasks. Such 'pre-existing software code' is known in the formal context as one or several APIs. The behaviour and utilization of APIs in a web software development hierarchy is similar to the use of 'modelling components' in systems engineering models. The thermal systems simulation software, TRNSYS (TESS, 2009), provides a good analogy to understanding the use of APIs. Like APIs, TRNSYS "Types" receive input conditions from the outputs of other Types, perform a computational function internally to themselves, and then submit generated outputs to Types that are further along in the simulation hierarchy. This process is repeated per user-defined time-step over a user-defined time series. One of the key advantages of TRNSYS is that regular users can focus on developing an overall systems model instead of undertaking bottom-up programming of each individual physical process carried out by a Type. Libraries of formally established Types have been developed previously by scientific experts and are readily available to TRNSYS users. The separation between general user and the Type/component developer in the example of TRNSYS is directly analogous to the creation of mashups by different types of web software developers, with general programmers able to utilize APIs developed by other experts.

A brief introduction to JavaScript

A mashup presented in this paper has been developed in JavaScript (JS), a widely-used programming language for web applications and programmatic web services. As not all readers may be familiar with JS and how it compares to other languages, a brief introduction is made here. JS is a high-level, interpreted programming language comparable to Python and Perl. The language emerged in the 1990s to become one of the three technical pillars of the web, alongside HTML and CSS. The difference between HTML, CSS, and JS is made evident by the three keywords that define their typical use. These are, respectively: content, style, and algorithms. Hypertext Markup Language (HTML) scripts contain written code describing a website's static content and structure. Cascading style sheets (CSS) scripts contain written code largely for describing and interpreting the visual style of a website. JS scripts contain written code for all forms of algorithmic processes a website may perform, including dynamic re-writing of HTML and CSS content. In the web's early days, compilation and rendering of HTML, CSS, and JS script would happen client-side, or in the user's browser. Hence, a user's own processing hardware would be used to interpret JS script into machine code and execute it accordingly. Today, complex JS-based programs, like Google Docs, are too sophisticated to be downloaded, interpreted and executed by each user's desktop PC or smartphone. Instead, large JS-based applications are interpreted and executed server-side, or remotely on web servers using one of several available JS runtime environments.

Prior development of virtual reality applications for the built environment

Virtual reality (VR) has been a sort of buzzword in the building science community since the mid-1990s when the first generation of VR technologies was developed and promoted. Already at that time, VR and the related technology known as augmented reality (AR), were identified as possible future tools to assist with building design, facilities management, and simulation of building occupants' experiences (Whyte, 2002; Walczak and Cellary, 2002; Leinonen and Khknen, 2000; Whyte et al., 2000). Research in this area appeared to peak and then decline in the early 2000s as it became a matter for commercial entities to realize the ideas and prototypes developed first in research (Freitas and Ruschel, 2013). In that time, two barriers became evident which eventually limited any significant use of VR in the construction industry (Greenwood et al., 2008): 1) the expertise required to develop a VR application, as well as the computational effort and costs required to manage one, was prohibitive; 2) the available graphics-rendering and VR technology of the time could not create a convincing user experience.

What is different today, and is the basis for this work, is that the second generation of commercially-developed VR-technology, which emerged around 2011-2013, has made it considerably easier and cheaper to realise convincing VR applications (Amer and Peralez, 2014). In the current era of VR, applications can be programmed using more commonly used and streamlined programming languages. The data driving these applications is now more cheaply processed and easily communicated used 'cloud services', driven by the same programming languages used for VR software development. Last, entire VR experiences can be generated and projected on an already-ubiquitously owned technology: the smartphone.

Case Study Building

Over 2014-2015, a 550 m^2 office space was constructed in Singapore to serve as a living laboratory of the '3for2 Beyond Efficiency' concept. The office space, known further in this paper as the *case study building*, serves as a research, development, and demonstration platform for technologies and building design measures that can achieve combined energy, space, and material savings in high-rise buildings. Highly instrumented with sensors, the case study building is also being simultaneously used to research new techniques for intelligent building control and data analysis. It is from research on the latter topic that this paper has emerged. An illustration of the case study building is provided in figure 1 and further information can be found in Schlueter et al. (2016).

Data collection and processing from the case study's building management system

At the most fundamental software level within the modern building, measured building sensor data is created by building management systems (BMS). A comprehensive BMS network regulates numerous sensors, actuators, and the networked devices that facilitate communication between each node. The main focus of a BMS is to control the systems in a building to keep occupants thermally and visually comfortable, and healthy. A secondary function of the BMS, though intrinsically linked to its main function, is the collection and communication of sensor data measurements. The data storage and analysis functionality of conventional BMS has improved over the years due to various database and network technology improvements.

Our case study building features a modern Siemens Desigo CC BMS. It interacts with over 1,000 sensors and control points that have been installed primarily to support the wide research aims of the project. These points include air temperature, humidity, and CO_2 sensors, temperature and flow sensors for chilled water, valve positions, energy meters, and many others. One of the features of the BMS in the targeted case study is that the BMS' proprietary data server is accessible in real-time through a web-portal accessible through a representational state transfer (REST) API, to be discussed in the following section. This simply-described product feature has a far-reaching implications, as its been the main technology factor enabling the wider work described below.

Mashup for visualizing and managing real-time building information in VR

Structure of the mashup and workflow

An example mashup demonstrating a VR-based building information management tool has been developed for this work, with its structure illustrated in figure 2. The mashup utilizes open-source APIs for VR rendering and data management, A-Frame and REST, which are described further below.

The connection of the APIs to each other, including the formatting and processing of visualized data, is programmed in Javascript. This is the main part of the mashup that has required in-house bottom-up development. Meteor.js serves as as the overall framework, or wrapper, for the mashup. It is used to launch the tool on a web server, making it accessible to users of normal web browsers.

Additional applications and software code required for this work have included the Google Street View app - for generating equirectangular images of interior spaces, and Python - for processing raw BMS data and pushing data to a cloud database.

Meteor.js: An open-source framework for developing, packaging, and deploying JavaScript applications

Meteor.js is an established framework unto which one can organize the various JS scripts and external JS libraries/APIs necessary to execute a large JS-based software program. It has developed a niche particularly for the development of web applications that require interaction and visualization of real-time data to users. Meteor.js apps are heavily integrated with MongoDB data servers, which manage the storage and transmission real-time data. MongoDB is an open-source document-oriented database program particularly applicable to JS-based web applications.

In programming Meteor.js apps, one can establish links between different APIs and custom user-defined JS scripts that call each other dynamically - such as a condition driven by real-time data. For example, let's propose that a Meteor.js app calls an API to pull real-time data from a 3rd-party web service - perhaps a website that generates the score of a football match in real-time. The app can create a dynamic link with this data feed in such a way that a subsequent algorithm is triggered only upon a change in value - i.e, a change in the match's score. A very simply

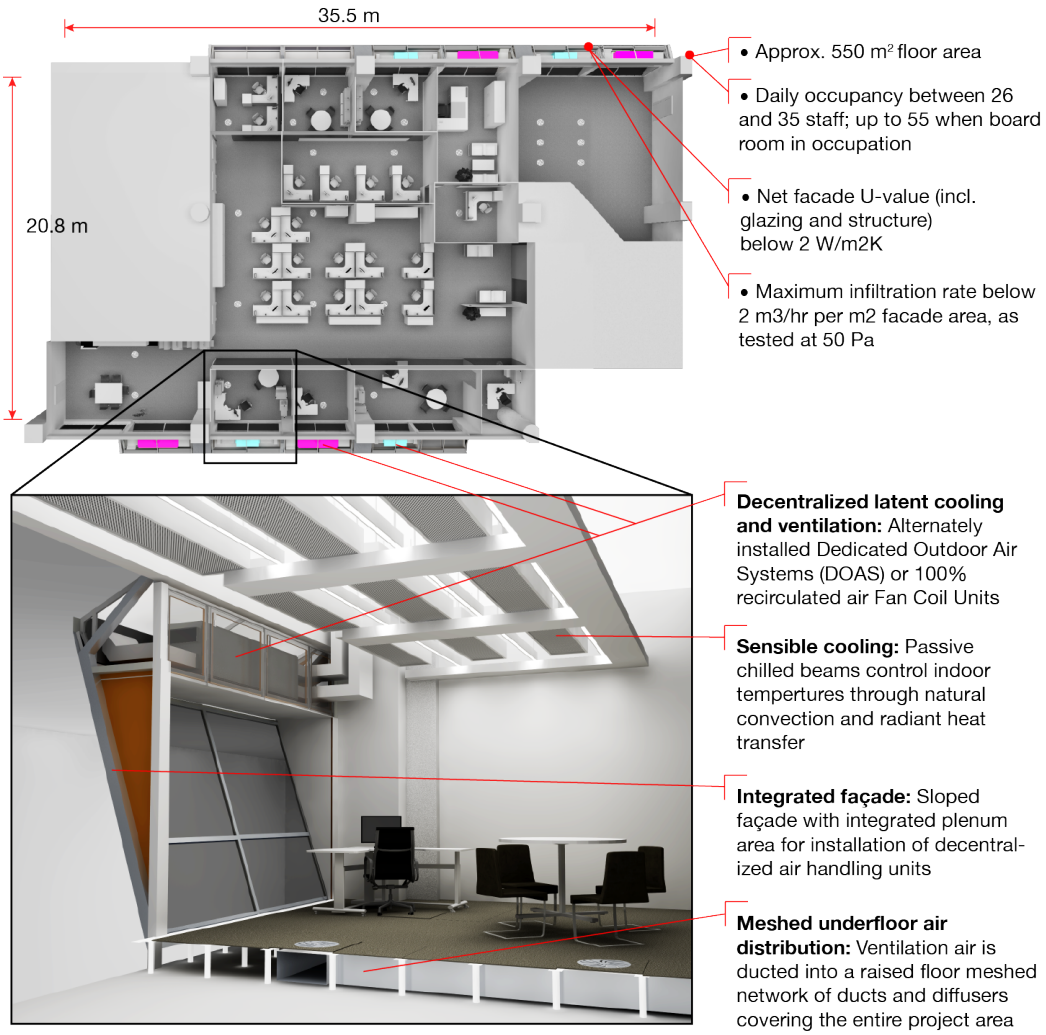


Figure 1: Overview of case study building: the 3for2 living laboratory in Singapore (from (Schlueter et al., 2016), with permission)

Meteor.js app based on this example could link the data feed API to a visualization API which updates a web browser-displayed data graph of the match's score when the score has changed. Again, the use of pre-existing APIs means that the programmer of this app would only need to code the links and conditions between the APIs, not necessarily the bottom-up program of the data feed and visualization.

For our current application, we are using Meteor.js in a very similar fashion as the example above. The management and visualization of data is largely handled by pre-existing APIs, with our programming code only establishing the dynamic links between these APIs and between the overall app and the web-browser. More information about Meteor.js, its origins and use, can be found in the book of Coleman and Greif (2015).

REST API for standardized data exchange between web services

One of the features of the BMS in the targeted case study is the aforementioned REST API that enables the real-time collection of sensor data through a web-services portal. This interface enables two-way interoperability between the case study's BMS and the internet. *REST* is a set of guidelines to facilitate the exchange of data using HTTP through a base URL and various methods such as *GET, PUT* and *POST*. While not a strict standard, *REST-ful* web interfaces have several common features which enable a knowledgeable programmer to quickly connect and extract information without prior experience (Pautasso et al., 2014). These features include programming architectural constraints that influence simplicity, performance and scalability. In the case study, a Python data extraction program executed on the BMS server queries measured data from the BMS at regular intervals, and posts this data to an InfluxDB cloud database via REST.

Time-series data management

InfluxDB is a database technology developed specifically for time-series data storage and management (InfluxData, 2016). This database is available

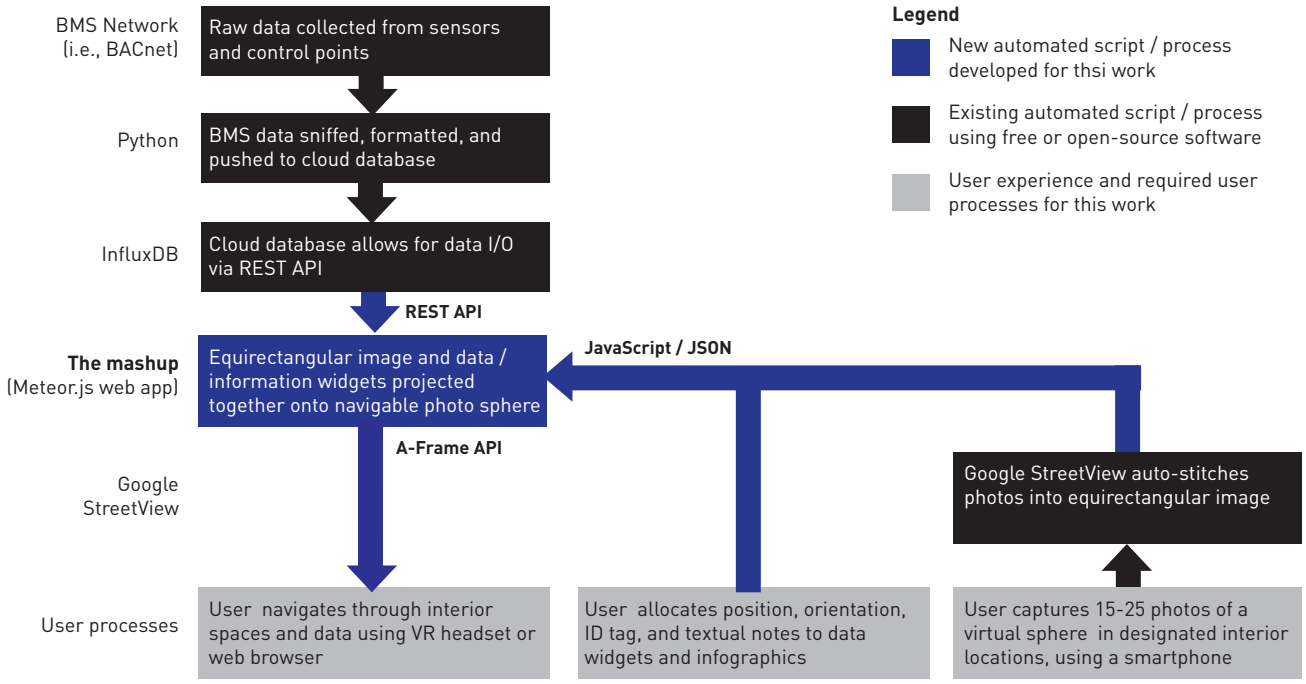


Figure 2: Overview of workflow; processes depicted in blue and black operate in real-time

through an MIT open source license and has no dependencies. It is purpose-built for temporal data and requires no special schema design for time-stamped data and numerous time-centric functions are available in its use. The InfluxDB database used in the prototype application, being updated in real-time, is called dynamically by the application's A-Frame API during rendering of VR scenes to a user's display.

A-Frame: API for the creation of cross-platform virtual reality applications

A-Frame is an open-source API for processing dynamic VR content on a web server, and rendering it in real-time within web browsers using conventional HTML (A-Frame, 2016). The capabilities of A-Frame far exceed the attributes used for this study, hence we will describe mainly the features of the API we have relied upon.

For the prototype application, in each rendered VR scene, the API receives as inputs:

- a static equirectangular image of the interior space to be rendered in VR
- a list and description of data and text widgets to display in the VR environment
- a unique data identification number that relates data widgets to their appropriate output from the REST API
- a list of coordinates indicating the desired locations the data and text widgets in the VR space

Linking the data widgets to the REST API is dynamic, hence the values displayed by the widgets in VR is updated in real-time.

Google Street View Application

Google Street View is a mobile application available on both the Android and iOS platforms. Using Google Street View, one can automatically generate equirectangular images of surrounding spaces using a smartphone's camera. 'Equirectangular images' refer to rasterized, composite 2D images of entire 3D photo spheres. They are a standard image format used for rendering static, real-image scenes in VR. As we assume that smartphones are owned by many, if not virtually all professionals in technical trades, we view Google Street View as an effective, fast, and ultimately free tool for generating spherical images of interior spaces in lieu of more advanced hardware. The VR scenes of our following case study have all been captured originally using iPhone6 smartphone and Google Street View version 2.6.1 (Google Inc., 2016).

Use of prototype tool

Creation of scenes and allocation of widget coordinates

As previously described, the generation of equirectangular images of interior spaces is handled automatically by the Street View app. Using the app, it takes approximately 1 to 1.5 minutes to generate a single equirectangular image of an interior space. The position and placement of data and information widgets in virtual interior spaces must currently be done manually by a user in a text-based format. In the current version of the mashup, a user must estimate by trial-and-error the Cartesian coordinates of each widget

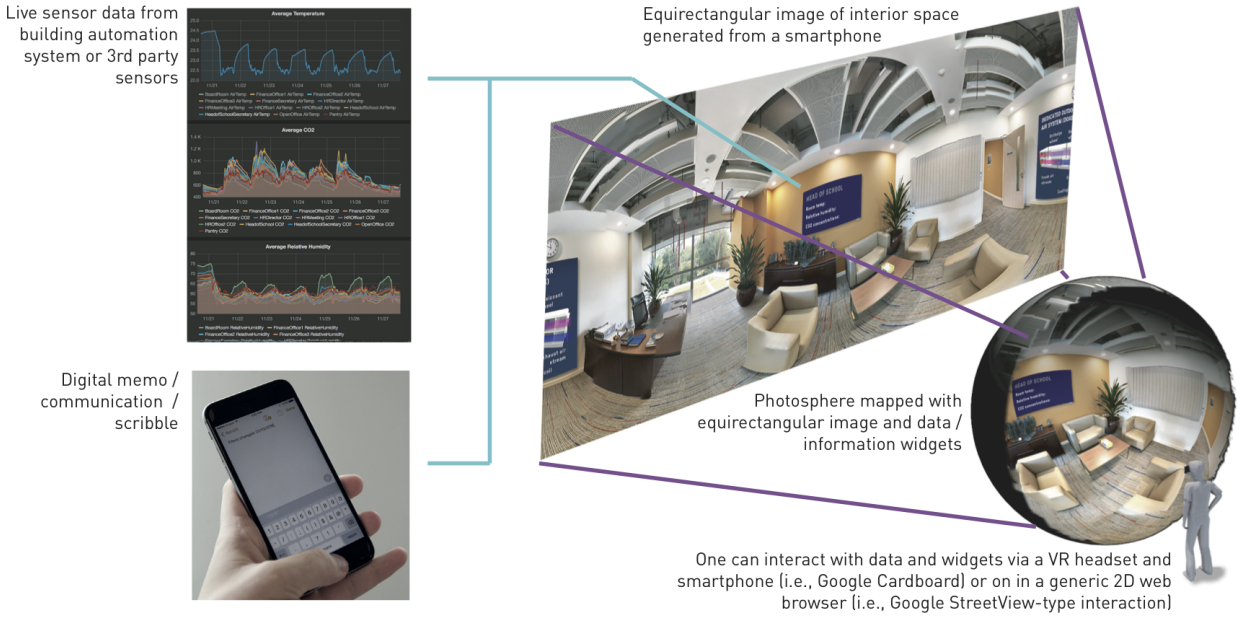


Figure 3: Overview of the VR application: Streamed data and prior stored information widgets overlaid onto an equirectangular image of a building interior

in relation to a defined origin point: the centroid of the virtual interior space. This must also be done ex-ante to the use of tool so that A-Frame can correctly position widgets on generation of a virtual scene.

Rendering and navigation between virtual spaces

The rendering of interior spaces is done automatically by the A-Frame API, with data widgets updated in real-time via the REST API. An illustration of the final tool's composition, in terms of the user experience, is provided in figure 3. With a VR headset, such as a Google Cardboard viewer, a user can interact with the space by rotating their head and examining their surroundings. As we have developed the tool with a button-less interface, users currently interact with the space based on the duration of time they look at a particular element on display. For instance, users can navigate between spaces by staring directly at floating spheres that depict a possible next destination. An example is shown in figure 4.a. It should be stated that a VR headset is not required to interact with the application. Desktop PC users can view the experience in a 2D web browser.

Example uses

At the time of writing, a prototype deployment of the application can be viewed online here: `http://3for2.fcl.ethz.ch/vr`. Figure 4 provides static images depicting key features of the prototype.

Discussion of results and future work

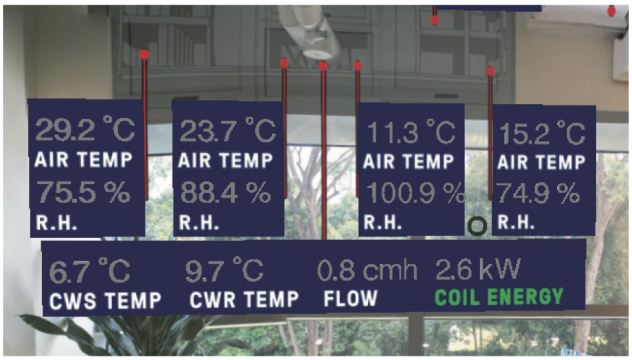
The simple prototype tool presented in this work, and illustrated by example in the previous section, currently offers capabilities comparable to present BIMs, such as documentation, data management, visualization, defect detection, indoor navigation, and identifying the location of building components. The main disadvantage of the new workflow is that, at this time, the user parameterization of data widget locations and orientations (currently specified in spherical coordinates) may be considered non-intuitive to non-specialist users. It would be more efficient and intuitive if users could interact with widgets directly in the VR environment, including editing, redrawing, and repositioning them.

These advantages and disadvantages are limited to the view of the authors of this study, however, and this in and of itself can be considered a limitation of this overall work. Testing of the proposed tool with a regular user base, preferably users of existing BIMs, is required in the future.

However, what became evident from the development of the prototype tool in this work was the ease at which an apparently powerful VR applications can be programmed and deployed. The prototype tool presented in the previous sections was produced by a single software programmer in 15 working days, working under the supervision of a team of building scientist researchers. This short duration of work must be put into context when comparing the capabilities of the prototype to large-scale commercially developed BIM programs. Similarly, this short duration effort provides evidence of the strength of JavaScript 'mashups'. Increasingly

a. Users navigate between interior spaces by focusing their center view on floating photo spheres and waiting ~5 seconds for the view to change. The photo spheres depict the image of the location to travel to.

b. The VR experience can be used for information sharing as much as data visualization. Arbitrary infographics can be assigned and updated based on new information. This can be useful for educational purposes (as shown) or issue tracking of matters relevent to facilities management.



c. The strength of the tool is most evident in visualizing data that is normally hidden. Here, live mechanical ventilation system data is displayed for each stage of an air handling unit's process. This air handling unit is located within the facade of an occupied space, hence the unit and data is normally hidden from view.

Figure 4: Use examples of prototype tool developed for case study building

powerful web-driven software tools can be developed quickly by leveraging existing application code in the open-source community.

Within the period of writing this paper, Google launched its successor to its Cardboard VR platform: Google Daydream (`https://vr.google.com/daydream/`). Daydream will allow users to interact with web-based VR environments using a low-cost, motion-tracking, hand-held dongle. The incorporation of a hand-held dongle into the prototype tool is seen to be a crucial next step. By replacing manual, written allocation of widget coordinates when configuring a VR interior space, a dongle could be used to allow a user to 'drag-and-drop' widgets in their desired location, as well a edit information widgets in the VR mode.

Conclusions

This paper has *not* set out to make a case that traditional BIM software is unnecessary for existing buildings. However, a newly-developed and tested workflow has been presented which indicates that existing APIs for virtual reality, data visualization and analytics may soon, if not already, allow software developers to produce BIM-like applications with considerably low effort.

Overall, we view it may serve the building simulation community well to keep a closer eye on the rapidity in which the open-source, cloud-based software community is developing. New tools for data management, processing, analytics, and visualization are increasingly offering features desired by the building performance simulation and analysis community. The programming code behind the prototype workflow presented in the paper will be made publicly available at the time of publication.

Acknowledgments

The authors would like to acknowledge Naor Biton, the staff of United World College South-East Asia, and the support of Siemens Building Technologies in the development of this work.

References

A-Frame (2016). A-Frame.

Amer, A. and P. Peralez (2014, October). Affordable altered perspectives: Making augmented and virtual reality technology accessible. In *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, pp. 603–608.

Azhar, S. (2011). Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry. *Leadership and Management in Engineering 11*(3), 241–252.

Coleman, T. and S. Greif (2015). *Discover Meteor: Building Real-Time Javascript Web Apps*. E-book.

Freitas, M. R. d. and R. C. Ruschel (2013). What is happening to virtual and augmented reality applied to architecture? In *18th International Conference on Computer-Aided Architecture Design Research in Asia (CAADRIA 2013)*, pp. 407–416.

Google Inc. (2016). Google Street View.

Greenwood, D., M. Horne, E. M. Thompson, C. M. Allwood, C. Wernemyr, and B. Westerdahl (2008, January). Strategic Perspectives on the Use of Virtual Reality within the Building Industries of Four Countries. *Architectural Engineering and Design Management 4*(2), 85–98.

Ilter, D. and E. Ergen (2015, July). BIM for building refurbishment and maintenance: current status and research directions. *Structural Survey 33*(3), 228–256.

InfluxData (2016). InfluxDB.

Jung, J., S. Hong, S. Jeong, S. Kim, H. Cho, S. Hong, and J. Heo (2014, June). Productive modeling for development of as-built BIM of existing indoor structures. *Automation in Construction 42*, 68–77.

Leinonen, J. and K. Khknen (2000). New construction management practice based on the virtual reality technology. In *Construction Congress VI: Building Together for a Better Tomorrow in an Increasingly Complex World*, pp. 1014–1022.

Miettinen, R. and S. Paavola (2014, July). Beyond the BIM utopia: Approaches to the development and implementation of building information modeling. *Automation in Construction 43*, 84–91.

Pautasso, C., E. Wilde, and R. Alarcon (Eds.) (2014). *REST: advanced research topics and practical applications*. New York, NY Heidelberg Dordrecht: Springer.

Schlueter, A., A. Rysanek, C. Miller, J. Pantelic, F. Meggers, M. Mast, M. Bruelisauer, and K. W. Chen (2016). 3for2: Realizing Spatial, Material, and Energy Savings through Integrated Design. *CTBUH Journal* (II).

Tang, P., D. Huber, B. Akinci, R. Lipman, and A. Lytle (2010, November). Automatic reconstruction of as-built building information models from laser-scanned point clouds: A review of related techniques. *Automation in Construction 19*(7), 829–843.

TESS (2009). TRNSYS 17.

Volk, R., J. Stengel, and F. Schultmann (2014, March). Building Information Modeling (BIM) for existing buildings Literature review and future needs. *Automation in Construction 38*, 109–127.

Walczak, K. and W. Cellary (2002). Building Database Applications of Virtual Reality with X-VRML. In *Proceedings of the Seventh International Conference on 3D Web Technology*, Web3D '02, New York, NY, USA, pp. 111–120. ACM.

Whyte, J. (2002). *Virtual Reality and the Built Environment*. Routledge.

Whyte, J., N. Bouchlaghem, A. Thorpe, and R. McCaffer (2000). From CAD to virtual reality: modelling approaches, data exchange and interactive 3d building design tools. *Automation in Construction 10*(1), 43–55.