A simplified software architecture for self-updating Building Information Models (BIM)

By

Pierre Fuller

B.S., Civil and Environmental Engineering, Magne Cum Laude (2007)
B.S., Architecture, Magne Cum Laude (2007)
Lawrence Technological University

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Civil and Environmental Engineering

at the

Massachusetts Institute of Technology

September 2009

Signature of Author_____
Department of Civil and Environmental Engineering
August 7, 2009

Certified by_____
Jerome J Connor
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by_____
Daniele Veneziano
Chairman, Departmental Committee for Graduate Students

A Simplified Software Architecture for Self-updating Building Information Models (BIM)

By

Pierre Fuller

Submitted to the Department of Civil and Environmental Engineering
on August 7, 2009 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Civil and Environmental Engineering

ABSTRACT

Building Information Modeling (BIM) is an emerging software technology that is revolutionizing the architecture, engineering, and construction (A/E/C) industry. BIM technology employs "object-based 3D models—containing the physical and functional characteristics of a facility—that serve as a repository for lifecycle information in an open, interoperable format" [1]. The major difference between BIM and Computer-Aided Design/Drafting (CADD) is that the former includes geometry and a plethora of building information while the latter includes only geometry. BIM utilization in the AEC industry has increased due to 1) BIM tools increasing productivity in design tasks; 2) the increasing number of private and government agencies that have instituted BIM requirements; 3) the pervasive use of computer analysis and simulations models; 4) the benefits of BIM as lifecycle management tool. Current literature shows trends of a transition from a "passive"—static model-based—approach to an "active"—dynamic model-based—approach. The active approach requires the integration of BIM with sensors to create "self-updating" building models.

Previous research introduces the concept of a self-updating building model ([2], [3], [4]). These systems involve complex software architecture and may perpetuate the problem of software interoperability. This thesis explores the following question: May a similar system be created to synthesize dynamic sensor data while improving upon previous research and simplifying the software architecture? The author describes a prototype system, called LiveBuild, which integrates commercial BIM software with other off-the-shelf software components to create a self-updating building model.

LiveBuild is the first self-updating building model that operates as an extension to existing commercial BIM software. Therefore, the transition from static to active building models is as simple as installing a plug-in. LiveBuild may serve as the basis for future research in self-updating building by providing simplified system that is well integrated with state-of-the art commercial design software. Likewise, the prototype is applicable for professional practice by allowing firms to use their existing BIM software to perform "pilot projects" with self-updating technology. The current prototype supports an interface with single commercial BIM software (Autodesk Revit 2009) product however future prototypes may extend both the functions and interfaces for other BIM software.

Thesis Supervisor: Jerome J. Connor
Title: Professor of Civil and Environmental Engineering

- This page intentionally left blank -

Acknowledgements

I would acknowledge my Lord and Savior Jesus Christ who is the head of my life. I would like to thank my mother, my family and Renèe for their prayers and continued support.

Finally, I would like to thank my advisor Prof. Connor for his guidance and my office mates Todd and Simon for their input and helpful criticism.

Thank you all!

He who began a good work in you will carry it on to completion until the day of Christ Jesus.

- Philippians 1:6 (NIV)

- This page intentionally left blank –

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1 What is BIM?

Building Information Modeling (BIM) is an emerging software technology that is revolutionizing the architecture, engineering, and construction (AEC) industry. BIM, and similar technology, has been successfully used in many fields however, the scope of this document is limited to BIM use in the AEC industry. BIM technology employs "object-based three-dimensional (3D) models—containing the physical and functional characteristics of a facility—that serve as a repository for lifecycle information in an open, interoperable format" [1]. The major advantage of BIM over typical Computer-Aided Design/Drafting (CADD) is that the former includes geometry and semantic building information while the latter includes only geometry (Figure 1). Semantic building information supplements the 3D model by describing the attributes and properties (e.g. material composition, mechanical properties, manufacturer, part/serial number, etc.) of each building component.

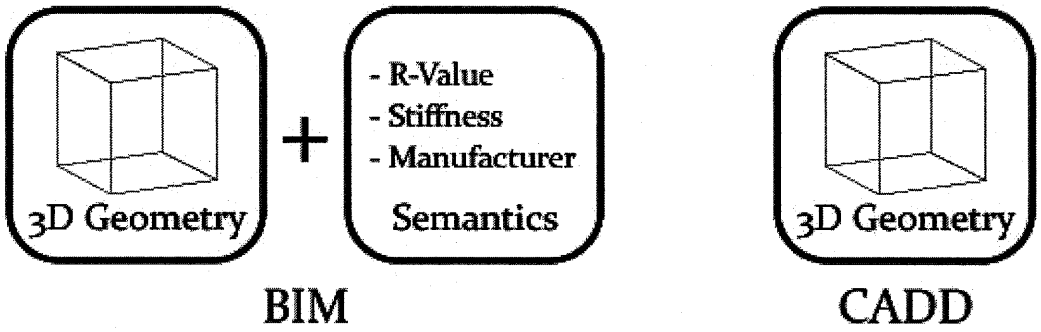

Figure 1: BIM vs. CADD

BIM also facilitates a level of electronic communication and collaboration that was not previously achieved in practice. The most successful implementation of BIM technology requires the involvement of project stakeholders (designers, contractors, owners, etc) early in the design process. Individual designers' BIM models (architect, structural engineer, mechanical engineers) may be leveraged as tools to communicate design decisions that impact other designers' decisions. Working in with BIM technology

incentivizes the early creation of a building model to initiate interdisciplinary collaboration earlier in the process where design decisions have the greatest potential impact on the final project.

1.2 BIM Growth Factors

Several major factors have accelerated the growth of BIM within the AEC Industry: 1) BIM tools increase productivity in common design tasks; 2) the large, and increasing, number of private organizations and governmental agencies that have formally instituted BIM requirements and standards; 3) the pervasive use of sophisticated analysis and simulations models and 4) the demonstrated and expected benefits of BIM as lifecycle management tool. A discussion of each contributing factor follows.

1.2.1 Increased productivity in design tasks

Building information modeling software, regardless of the specific developer, has features that may immediately improve productivity for designers. These features include coordination and collaboration tools, parametric design capabilities and the automated creating of two-dimensional (2D) design drawings. BIM offers improved tools for collaboration between designers. For example, a team of various design professionals all working in BIM technology may combine their individual models to identify "interferences" or "clashes" between building components, a task not easily achieved with CADD. Parametric tools establish relationships between model components. With these relationships in place, a change in one parameter may automatically modify many model components based on the predefined relationships. For example, if the spacing between structural joists were parametrically related to the span of the joists, a change in the span would result in an automatic change in the spacing between joists. Parametric tools improve flexibility especially in early design phases where many options may be explored. BIM software also automatically extracts 2D drawings from the 3D model. Many design firms have invested in BIM technology to take advantage of improved interdisciplinary coordination, design productivity and document consistency.

1.2.2 Institution of BIM requirements and standards

There are an increasing number of private organizations and governmental agencies that have formally instituted BIM requirements and standards. Many large design firms are making drastic transitions from CADD to BIM. Furthermore, many governmental agencies are requiring the use of BIM on all future projects.

Design firms are investing significant money, time, and effort in BIM technology. Many companies maintain one license for each individual on the design staff. A single license of BIM software can range from 10 - 60% the annual salary of the average design personnel and the time required for software training drives the cost even higher. Many firms have made drastic shifts across their entire company from a CAD-based to a BIM-based design process, making BIM their primary design and documentation tool. Many companies have set formal deadlines, after which they will complete nearly 100% of their work in BIM software [5].

Governmental organizations, who are also large clients to design firms, have also benefited from proven costs savings on project budgets. Virtual collaboration and clash detection in BIM, results in fewer unexpected problems on-site which, in turn, results in fewer change orders, reduced overall cost, and more reliable construction schedules. Many owners now *require* the use of BIM software in project design and delivery. The US General Services Administration (GSA) is responsible for providing building and space for all federal agencies. They are responsible for the developing and administering design standards for new projects and management of existing facilities. The GSA states that

> *...all major projects that receive design funding in [the 2007 financial year] and beyond are required to submit a spatial program [building information model] to GSA prior to final concept presentation. GSA design teams use BIM to validate spatial program requirements (e.g., area, efficiency ratios) more accurately and quickly than traditional 2D approaches.*[6]

Other large facility owners like the US Coast Guard, Texas Facilities Commission [7], State of Wisconsin Department of State Facilities [8], General Motors, Department of Veterans Affairs [9] have all embraced BIM technology. Owners have clearly shown their confidence in and commitment to the use of BIM technology and as a result BIM has seen rapid growth throughout the building industry [10]. From the top down, the AEC industry is invested in BIM.

1.2.3 Pervasive use of analysis and simulations models

Analysis and simulations models may cover a variety of mechanical or passive (airflow, fire, natural light, etc.) building systems. The performance of each building system may be simulated to help owners determine the most efficient and effective systems. BIM models can serve as the basis for simulation models. Information from BIM may be "imported" into simulation software instead of recreating the entire model. Designing for sustainability often requires computational models to simulate the expected performance of a system and optimize its performance to reduce financial and environmental costs during operation. Sustainable design is loosely defined as "[avoiding] resource depletion of energy, water, and raw materials; [preventing] environmental degradation caused by facilities and infrastructure throughout their life cycle; and [creating] built environments that are livable, comfortable, safe and productive" [11]. The United States Green Building Council's (USGBC) has championed much of the industry-accepted standards for sustainable design. The LEED green building rating system developed by USGBS, is intended to provide "building owners and operators a concise framework for identifying and implementing practical and measurable green building design, construction, operations and maintenance solutions" [12]. USGBS also verifies and certifies buildings that meet LEED standards. It is important to note that the current LEED standards are focused on checking the "design" performance and do not verify the "actual" performance after the facility is in operation. The adoption of LEED by Federal, State and local government has helped it to become the leading standard for sustainable design in the US. Moreover, the influence of sustainability has increased awareness of lifecycle cost

analysis. Now, many owners are found that the lifecycle costing is a more accurate than traditional estimation methods (e.g. initial cost only).

1.2.4 BIM in lifecycle management

By definition, BIM is a "repository of lifecycle information" and should therefore provide benefits throughout the building lifecycle (design to demolition). Currently, the benefits of BIM are only seen in the design and construction phases. A 2007 survey conducted by Stanford's Center for Integrated Facilities Engineering (CIFE) found that though 50 to 60 percent of the respondents offered pre-project planning and construction management services, they were less likely to use BIM (called VDC or Virtual Design and Construction in the CIFE report) in these phases than they were in the design and documentation phases [10]. Furthermore, BIM use in planning and construction management phases shows little growth between 2006 and 2007 while the areas of conceptual design, design development and design documentation have seen large growth.

With such a large industry commitment to BIM technology there will undoubtedly be future use in other lifecycle phases. Building operation requires the largest financial investment of all other building phases and therefore offers the largest potential cost savings. In the future, architects and engineers may find ways to offer BIM-based operations and maintenance support to extend their services (and fees) throughout the building's lifecycle. The under serviced phases (pre-planning, construction management, operations & managements) will continue to grow not only in total number but will also grow relative to the design phase because: 1) Owners will be better informed of building technology and 2) Research will continue to advance the technology operations and management; 3) the aforementioned growth factors will continue to influence BIM adoption throughout the industry.

1.3 Limitations

BIM technology has several technical, legal and practical limitations. The largest technical challenge is the lack of interoperability standards for BIM file formats. The

complete exchange of information between BIM software has eluded the AEC industry. Legal challenges include concerns over design liability and BIM model ownership (e.g. Who is responsible for integrating the final model from all designers?). The practical limitations include the ability to manage an appropriate level BIM model detail, general lack of knowledge and uncertainty about the future of BIM technology, and prohibitive cost.

1.4 Future of BIM

Future BIM systems must not only support the existing functions of BIM but must also facilitate integration with other technology and new data types. The nature of the information currently contained in building models is static. However, when objects have dynamic attributes (e.g. varying temperature) it is not longer appropriate to maintain only static information. Future BIM models must support not only static information but also dynamic information. Sensors may supply real-time information about building components to the model. Current computer-aided facilities management (CAFM) software contains drawings (floor plans, details, etc.) and spatial management tools (management and allocation of space to various functions). While extremely useful to owners of large facilities, this is a relative low-level use of building information. Future BIM systems will continue to support aspects of the current CAFM software but will also develop a framework to support the current trend automated and sustainable systems. Spatial management and building maintenance activities will benefit from continued support of static information while future systems will require benefit from real-time functionality.

1.5 Sensor Technology

The research presented in this study is not limited to any single type or group of sensors. The major difference between various sensors is the type and amount of data produced (e.g. sample rate, size of the data, and the type of data) but at a high-level that are all data producers. The prototype system proposed in this document (see chapter 3 LiveBuild

Prototype), is designed to capture sensor data regardless of the sensor type. Therefore, this study does not provide a review of sensor technology.

Chapter 2: Computing in the Building process

2.1 Introduction

This document investigates the current and future use of computing in each of the design phases. Therefore it is appropriate to first describe each of the design phases with and without the influence of BIM. Each building project is unique and may include several phases of development. The phases may differ depending on the team in charge of developing the project, or the project delivery method (design-bid-build, design-build, etc). However, it is common for capital building projects in the United States to include the following four phases: planning, design, construction and operation. Most of the "building phases" are made up of sub-phases.

The following description of building phases represents the process for capital building projects designed and built within the United States. This assumes that the client has, of course, made the decision to construct a new building. The determination of the need for a new building may be regarded as a design phase but is not included here because it has little relevance to the use of BIM modeling. The decision to undertake a new project is made by the owner with limited input from design professionals; it is not an appropriate phase for the utilization of BIM.

2.2 Building Phases (without BIM)

2.2.1 Pre-design planning/programming

The planning process is dependent upon the needs of the client. In the preplanning phase, the client/owner has already determined the need for a building project and has hired at least one design professional—usually an Architect—to begin the process. Project requirements and specifications are defined during pre-design and will later facilitate the design phase. Most clients do not have advanced knowledge of the building process and lack the expertise to articulate detailed, technical requirements of the building. Therefore the designers must establish, through systematic interrogation, the

specifications of the building to satisfy the clients' needs. This results in the enumeration of all spaces that should be included in the final design; the type and size (area) and nature of use for each.

2.2.2 Design

In conceptual design, designers generate many design ideas that satisfy the client's requirements, as determined in pre-planning. Designers may generate several significantly different building "concepts." At this time, designers brainstorm ideas, determine feasible approaches and may eliminate design concepts that are not feasible. Conceptual designs are presented and discussed with the building owner. The designs have a sufficient level of detail to provide a sense of what the designers would like to achieve in the final design but does not represent the level of detail that would be present in a complete design. "Back-of-the-envelope" approximations are made to maintain a level of reality but the specifics are not well articulated at this point. This process may be iterated until the client and the designers are satisfied with the conceptual design. Eventually, the owner with the counsel of the design team will approve a design concept [13].

Design development commences upon the selection of a single concept design for the building. Designers focus on the detailed design of all project components. "Back-of-the-envelope" calculations are replaced by specific design parameters from design codes.

Design documentation or document development includes preparing all the drawings that convey the designer's intent. Detailed drawings with dimensions, specifications and performance requirements are also completed during this phase.

This design procedure, from concept to document production, is typical of many design professionals. Architects, structural engineers, mechanical engineers, and other designers will each perform a similar process on their portion of the design.

2.2.3 Construction

For bidding, the nearly finalized construction documents (often called the bid documents) are "released" for contractors to bid on them. "Bids" are an estimate of the cost that a contractor— the general contractor or GC—will charge the owner to construct a building. The owner may offer a public or private bid. The former is open to the public; the latter is only open to a preselected list of contractors. The contractors create a bid by identifying the materials and equipment needed to construct the building as shown on the bid documents. Contractors use a variety of methods to construct detailed estimates that account not only for what is shown on the bid documents but also other parts needed for construction. Construction cost references, estimating software and detailed spreadsheets are several estimation methods used. Regardless of the estimation method, the estimate is usually overseen by an experienced estimator to verify the accuracy and completeness of the bid. Contractors may also collect bids from various subcontractors, who performed specialized construction, for inclusion in their final bid to the owner. In the US the contractor with the lowest bid, or nearly the lowest bid, is often awarded the construction contract.

The construction activities include all activities necessary to construct a facility and turn it over to the owner. The tasks include all phases of construction from site preparation, earthwork & excavation and the actual building construction. Contractors are responsible for constructing the foundation, main structure, building enclosure, interior wall partitions, mechanical systems, permanent equipment and finishes. The completion of these tasks are performed by contractors skilled in various areas including, steel erection, concrete, masonry, plumbing, electrical, etc.

The construction site requires the coordination of construction equipment and building materials. These onsite logistics are often loosely included in the construction documents; however the detailed onsite logistics is determined by the general contractor or construction manager. Most management methods use 2D drawings to plan the use of space and equipment onsite. Material storage areas or "lay-down" areas may initially be

organized but quickly deteriorates into an unorganized area. For very large projects (with many parts requiring onsite storage) attempts are made to create a plan for material placement to facilitate easy retrieval later.

2.2.4 Building Operation & Maintenance

Operation and maintenance (O&M) begins when the building is handed over to the owner. The owner is responsible for maintaining all the physical building (structure, façade, foundation) and the building systems (mechanical, electrical, plumbing, etc.). Some owners use computer aided facilities management (CAFM) tools to help coordinate building maintenance.

2.3 "Over-the-wall" design paradigm

The AEC industry is undergoing a dramatic shift from fragmented design teams to integrated design teams that innovate through collaboration. Traditionally, AEC design projects are delivered in a workflow called design-bid-build (DBB). In DBB the owner secures separate contracts with individual designers and contractors. In contrast, in another workflow called Design-Build the contracts for each are combined; the designers and contractors usually work as a predetermined team [14]. DBB segments the design and construction processes into discrete phases where the design is often be completed or near completion before contractors are involved. Furthermore, design professionals have traditionally worked independently to complete their work scope before involving the next design professional. The result is a so called "Over-the-wall" design approach where each designer completes their design and then passes it over the wall to the next designer; the work of each designer is completed independently from all others (Figure 2).

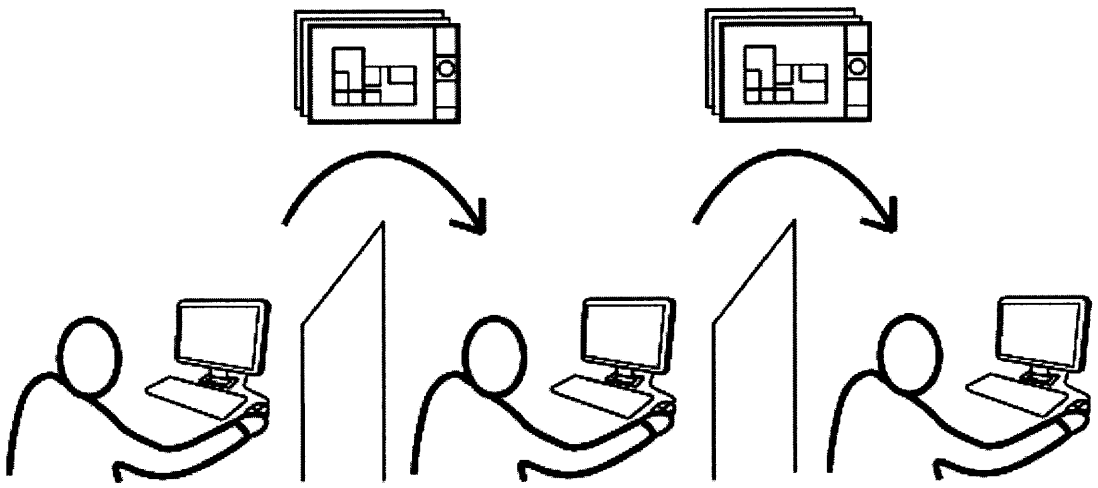

Figure 2: Traditional over-the-wall workflow (or design silos)

After each has been completed, a final check is performed to identify and resolve interferences between design components. The use of simple CAD tools would facilitate more collaboration that that illustrated in this model. With the increased in popularity of Design-Build, many owners and designers have recognized the benefit early interaction between various designers and have abandoned the over-the-wall approach. If one end of the workflow spectrum is "over-the-wall" then the opposite end is a BIM workflow: collaborative from the beginning and to end.

2.4 Building Phases (BIM integrated)

2.4.1 Pre-design planning/programming

BIM modeling generally does not occur when defining the requirements. However, once the requirements are defined they may be maintained in a database or verification software to facilitate model checking later (See Sec. 2.4.2 Design; Subsection "Design Checking").

2.4.2 Design

Conceptual design

Visualization. Visualization is an important aspect of the conceptual design process because it conveys the design idea to the client. BIM models may be fully rendered to for the presentation of static snapshots or a real-time "walk-through" of a design concept.

Parametric Design. The parametric features reduce the time and effort to create variations of design concepts. This allows designers to easily create options to present to clients to get feedback. For example, a single design concept with several parametric variations may be presented to a client. Each variation would help the client to understand the tradeoff of between design options, function, aesthetic and cost.

Cost analysis. Conceptual design cost analysis is approximated by "rule-of-thumb" or estimates based on the building type and unit cost (dollars per square foot) [15]. This estimation method does not precisely represent the cost of the final project; however it does present a rational means of comparing the relative cost of one design concept to another. Using the BIM model as the basis of the conceptual estimate will reduce the time and manual effort for early estimation. Finally, as conceptual design culminates in the selection of a single design concept, the conceptual BIM model may, at least in part, serve as the basis for the design model used in design development.

Design Development

Collaboration. BIM facilitates collaboration by the design team early in the process so that input from other design consultants can be considered and included at a time when it may still be able to influence the initial design [15]. Design models are also exchanged between design consultants and combined to check for interferences (commonly referred to interference checking called clash detection). For example, HVAC ducts commonly have to be placed to avoid structural components. Integrating the various models may reveal an HVAC duct that unintentionally clashes with a structural beam. These interferences can be detected and remediated before construction begins, reducing the number of unexpected modifications and ultimately reducing construction costs.

Analysis and simulation. Analysis is an integral part of the design process for many engineers and design professionals. Moreover, energy modeling and building system simulations are often used to predict a building's energy consumption. BIM models may serve as the basis for analysis and simulation models. Using BIM models reduces the amount of additional modeling required to generate an analysis or simulation model. Several software developers have established "links" to third-party software. Other developers have included analysis and simulation tools in the BIM software. Regardless of the avenue, the BIM model may eliminate redundant modeling for various design, analysis, and simulation tasks.

Design Checking. BIM models also may be used for automated code checking or requirements verification. As cited in Section 1.2.2 of this document, the General Services Administration (GSA) uses BIM to verify that the final design concept meets their spatial requirements for the project. The BIM-based approach allows validation "more accurately and quickly than traditional 2D approaches" [6].

Document development

Automated drawing production. The primary benefit of BIM in document development is automated document production and coordination. In BIM software, buildings are "modeled" using 3D components instead of drafted line-by-line as in CADD systems. As a result of having a 3D model, all 2D drawings (plans, elevations, sections, etc.) are automatically generated by the BIM software. The largest fraction of time in the design process is typically spent on document production. BIM will help to shift time and effort from document production to design which could ultimately improve the quality of the final building design.

2.4.3 Construction

Bidding

Estimating. Much of the cost estimating may be automated by the BIM model but an experienced estimator should still manage the entire process. Estimators may then be

able to focus on more detailed estimating (i.e. at a level of detail that is not included in the BIM model).

Construction activities

Constructability. BIM models may be used to perform constructability studies. These studies may include large construction equipment or temporary construction elements in the BIM model to investigate the spatial limitations that will be encountered onsite. The investigation may be as broad as determining site accessibility for construction equipment or may be as detailed as identifying challenging construction tasks for construction workers. BIM models may provide information directly to the construction crews responsible for completing work in challenging areas [15]. Crews that understand the onsite challenges can better pre-plan their work to accommodate the challenges.

Scheduling. If information is known about the rate of construction for different components, construction crews can be managed on a detailed level to optimize their work efforts using what is called 'lines of balance schedule analysis' [15]. Also, visualization is also useful during construction. The BIM models may be used to visualize the construction schedule and to communicate the expected building progression to clients.

2.4.4 Operations & Maintenance

Controls. A detailed BIM model may be combined with other tools (simulations, sophisticated algorithms, analysis) to automatically control building systems. For operations control and process controls in building operation, "object model-based management is still quite new, and there will undoubtedly be vast improvements in the next few years"[15]. This quote by Kymmell (2008) highlights the need for continued research in the area model-based building control. The prototype system presented in this document addresses this need by improving upon the current self-updating building model research and making the technology accessible to more project stakeholders (designers, contractors, and eventually owners).

Chapter 3: LiveBuild Prototype

3.1 Research Question

The goal of the study is to build on recent developments in commercial software development to create a system that integrates building information modeling and real-time (or pseudo real-time) sensor data acquisition. Previous research seeks a similar goal of integrating BIM with sensors but involves complex software architecture (See [2], [3], [4]). This thesis explores the following question: May a similar system be created to synthesize dynamic sensor data while improving upon previous research and simplifying the software architecture?

3.2 Background

Building Information Modeling has only recently been adopted by building design practitioners (professional engineers, architects, etc). However, BIM technology draws from over 30 years of "building product model" research that defined a conceptual framework for a data schema—ways to represent BIM data—that would describe a building's physical components and functional attributes [16], [17], [18], [19], [20]. The building product model concept is well-research and established.

To date, several domain-specific data schemas have been developed including the CIS/2 data schema for Structural Steel and gbXML for energy/sustainability analysis, to name a few. Moreover, there are developer-specific schemas for each commercially available BIM software package. In addition, BuildSmart (formerly International Alliance for Interoperability, IAI) has also led efforts to define a general data-schema—called the Industry Foundation Class or IFC—that allows communication among all other schemas [21]. The fragmented nature of the building design and data schemas lead the National Institute of Building Sciences to introduce standards for BIM, called the National BIM Standard [1]. The standardization focuses on two general areas: Information and Process.

- *Information*: The BIM Standard establishes preliminary standards toward achieving interoperability, consistent information storing/sharing and information assurance.
- *Process*: Provides standards for authoring, reviewing and publishing BIM models in a collaborative process.

BIM models are intended to be an information repository throughout the lifecycle of a building including the concept, design, construction, operation and deconstruction phases. Standardization is needed to allow information to be exchanged between the phases of a building's lifecycle.

In construction, sensors have been used to track equipment on construction sites, providing simplified location tracking and retrieval of building materials on crowded, often unorganized construction sites [22], [23]. Song et al propose a system using off-the-shelf RFID technology for construction tracking with an expected cost that is less than other existing location sensing approaches [22]. The authors show that the research area of materials management has great potential for improvement and cost savings. To address this potential, the authors propose a construction tracking system that uses a roving RFID tag reader equipped with GPS. The position of the RFID tag reader is determined by the GPS and the tag reader in turn determines the relative distance to tagged objects (e.g. building materials). Field tests were performed to verify the operation of the system and to determine performance measures to compare the system to other real-time location systems. The GPS equipped RFID reader calculates the tag location with an error of (+/- 3.7m (+/- 3.6m for the RFID tag reader w/o GPS) 68% of the time and +/- 4.9m (+/- 4.8m for the RFID tag reader w/o GPS) 94% of the time. This study shows the use of sensing and computing extended beyond the design phase and is now being applied in construction. This is indicative of a trend to increase the use of computing throughout the building lifecycle.

Current BIM research also investigates integrating building models with real-time data—collected from sensors—to automate building systems. The concept of context aware

computing was introduced earlier in other fields of computing research [24], [25] and was later applied in the building industry [2].

Icolglu et al. (2004) proposes a distributed location sensing platform using the visual TRIP tag system developed by Ipina et al (2002) [26], [27]. The proposed system uses off-the-shelf camera and inexpensive, printed TRIP tags to locate objects. The system implements the Distributed Component Object Model (DCOM) protocol to allow distributed communication over a network. The four basic system components are: the Application Sever which manages the use of all system resources, including cameras; the Database Server which stores data on all the system's components in XML format; a web-based User Interface Server which allows human visualization, retrieval and modification of the system data; and TRIP clients which process the images with an algorithm to extract the presence, orientation, relative distance and identity of tags. Images captured by the cameras are sent via HTTP to TRIP Clients for processing. Data extracted by individual TRIP Clients is sent to the application server where it is aggregated with data from other clients and supplemental information regarding camera position. The Application Sever sends data to the Database Server for persistent storage. This research presents well structured software architecture for a real-time building system. The primary drawback of the system is that the TRIP based system requires line-of-sight to recognize objects; it is best suited to an environment with few visual obstructions.

Brunner and Mahdavi (2005) propose a software architecture for a self-updating life-cycle building model [3], [4]. The basis of the model is what the authors call the Shared Object Model (SOM) (Figure 3). The SOM Objects combine static information about an object with data from one or several sensors. The description of a building space or domain may require the aggregation of several SOM Objects in parent-child relationships.

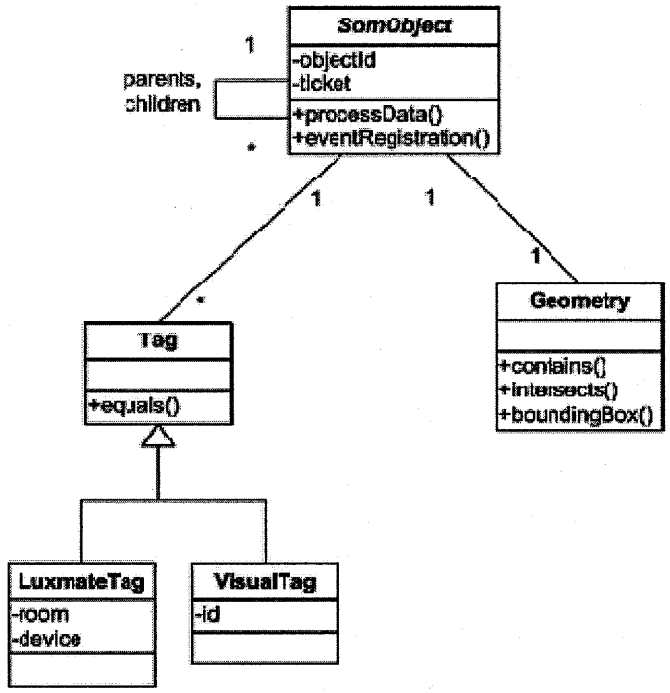

Figure 3: Shared Object Model (SOM)
Image Source: [4]

The Model Service combines an SOM-based building description with software interfaces for database storage, human visualization, sensor data collection and building simulations (Figure 4). The interfaces for sensor data collection and building simulations are facilitated by JavaSpaces, an implementation of Tuplespaces (Figure 5). JavaSpaces allows values to be passed into an intermediate "space" in tuples—a sequence of numbers with known data types—and are recognized upon retrieval by their signature.

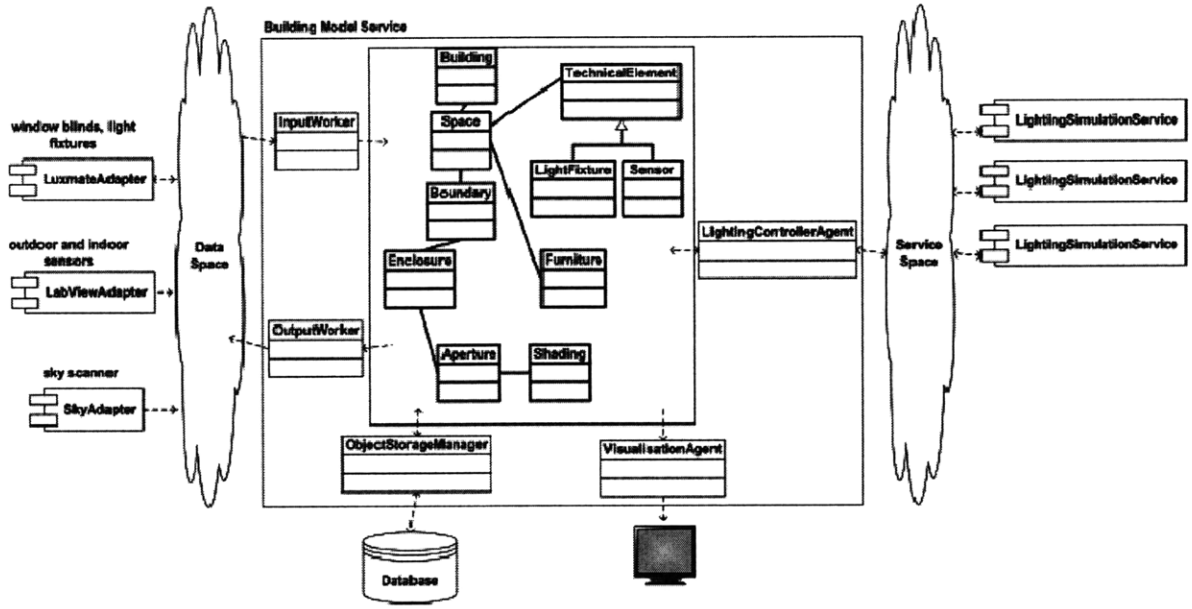

Figure 4: Overview of building model service
Image Source: [4]

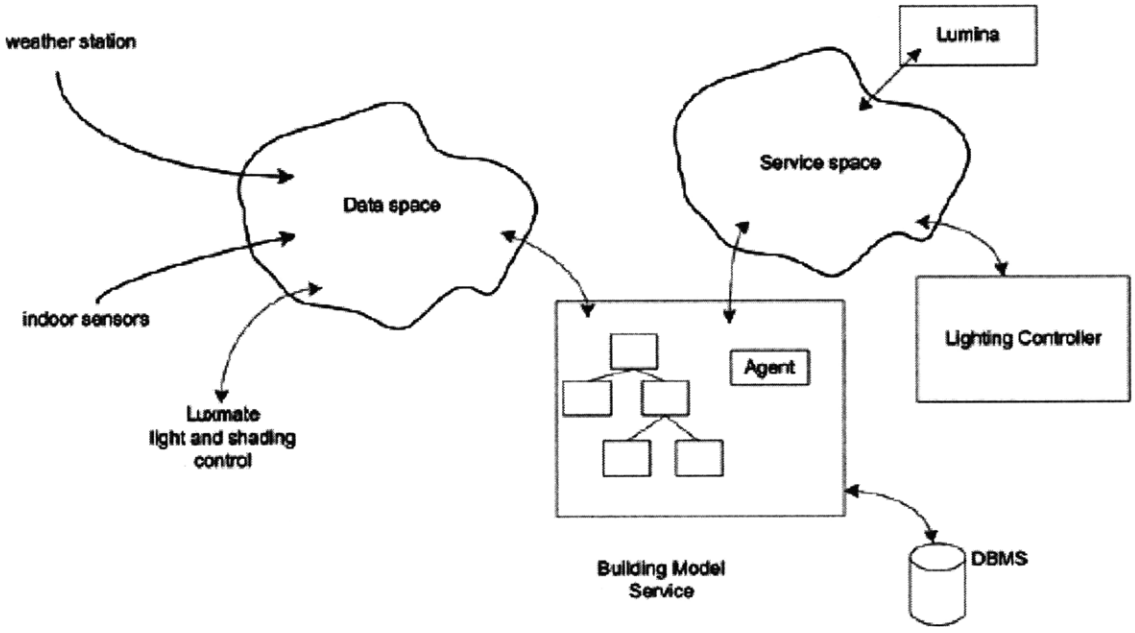

Figure 5: Software Communication via JavaSpaces (labeled Data space and Service Space)

Image Source: [3]

Brunner and Mahadavi's self-aware building model also utilizes simulations that determine the control regime of several passive and active lighting devices to maintain specified indoor light levels. The building model gathers information from sensors about ambient light level and the state of lighting devices. The model information is used to

simulate possible changes to the configuration of lighting devices. The simulation results of several possible configurations are used to select the most appropriate device configuration to maintain the light level.

Sharples et al. proposes a non-deterministic intelligent building control scheme [28]. Instead of using simulations, this system uses machine learning to "observe" occupants' and then tailor the automatic control of the system to predict a users' preferences. Sensors collect information on building spaces and the configuration of building controls. Using these as inputs, machine-learning may "train" itself to respond and modify building system controls in a similar fashion to the typical user. Predefined (deterministic) building control strategies are only used in case of emergency.

Research into the integration of BIM and sensors has been ongoing before the widespread availability of commercial BIM software. Therefore, existing research-based prototype systems are largely custom software architecture that is implemented in upon a "standard" file format [26], [3]. These systems are robust and allow domain-specific flexibility but have complex system architecture. The complexities include software-agents and intermediate programming languages to allow asynchronous communication between components. However, commercially available BIM software now provides a platform for a simplified integration of BIM and real-time sensing. This research asserts that commercial BIM software may be extended to include real-time sensor information and real-time decision-making.

3.2.1 Requirements for self-updating model

A prototype system designed to explore the integration of BIM and real-time sensor information should meet the functional and practical requirements listed below. The requirements were determined after a review of the current literature. Some requirements listed below are comparable to the requirements proposed in previous literature [3] but also include several new requirements that are unique to this study that improve upon previous research.

Support Sensor Data & Relationships. This is the basic function required to support a real-time building model. The prototype system should allow building components to be linked to sensors that collect physical data about the components (Figure 6). All BIM technology shares a common attribute, namely an underlying database of building information. The prototype system should supplement the content of this database with sensor data (and maintain the relationship of BIM-Objects to sensors).

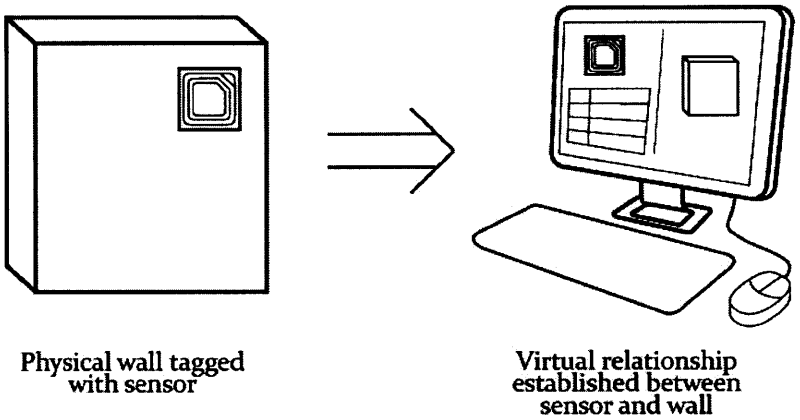

Figure 6: Establish link between sensors and objects

Support User/Data Interaction. Building data is stored in database tables that are arranged in a manner well-suited to computer interpretation; however, large database tables are not always easily understood by humans. Building data should be presented in a human understandable format which not only allows retrieval of existing data but also allow the addition of new data. The interface should support the functions (discussed below) for managing a self-updating building model. It should also be accessible to a wide range of users. The interface should be easily understood by individuals of varying technical backgrounds.

Operate in the Native Environment. The "native" environment is the software environment in which the building model is authored. It is important to operate within the native environment because moving out of the native environment is commonly achieved by using intermediate or common file formats like the IFC file format. Information exchange formats are often unreliable because various software vendors may

classify the same information in different ways within the same file type; this is the common problem of interoperability. The systems proposed in previous research first translate the building model into a new data schema, via an intermediate file format, before including sensor information. Sensor data integration should occur within the native environment to avoid the problem of information exchange. Maintaining the building model's original data schema does not solve the longstanding problem of interoperability however, unlike other systems it does not exacerbate the problem by introducing a new data schema. For example, Brunner and Mahdavi's proposed system first translate the building model into the SOM schema. An alternative system, like the one proposed by the author, preserves the informational structure of the original building model and supplements it with the information needed for real time sensing.

Support Various Building System Control. Another fundamental control that the prototype self-aware system should have is the ability to automatically actuate building controls. This allows the building to not only be self-aware but also self-regulating. Modern buildings contain many mechanical and electrical systems. The prototype system should provide an interface capable of actuating current building systems and provide the flexibility to incorporate future building systems.

Maintain central persistent storage. Persistent data storage (or non-volatile storage) refers to devices where data is maintained even when the power is lost. Central storage, as opposed to distributed, maintains a single location to store data. Persistent storage will accomplish two primary goals 1) maintain consistent building model representation 2) ensure the stability of the system in the event of power loss. A single, central data source will manage information consistency throughout a network of users. If the information is distributed among several individual machines then all these machines must remain "online" for the data to be accessible by others clients in the network. A central storage location allows individuals to remotely modify the building model while maintaining consistent information to all other users in the network.

Manage Concurrency. The prototype system should properly handle concurrent read and write functions on an individual object (Figure 7). For example, a sensor in the process of sending data to the building model at the same time a sensor reading is requested by another client should not compromise the behavior of the system.

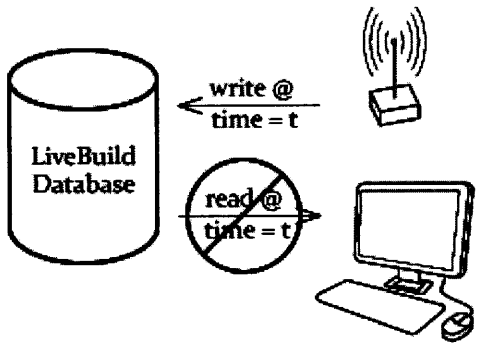

Figure 7: Example concurrency management

Limit Latency. Latency is the time it takes for a sensor reading sent to the database to become available within the model. High latency is problematic for systems that must dynamically respond to "real-time" or near real-time changes. The latency should not be prohibitively high and minimizing latency is ideal.

3.3 LiveBuild Prototype System

The author developed a prototype self-updating system, called LiveBuild short for "Live Building" that improves upon previous research and simplifies the software architecture. It uses off-the-shelf BIM software—with no real-time functionality—and modifies it to respond to real-time data in its native environment and dynamically control building systems. This tool will later be used as a platform on which to build real-time decision-making systems.

Several software developers offer commercial BIM software for the AEC industry including Autodesk, Bentley Systems, Ghery Technologies, Graphisoft and Tekla to name a few. An overview of the strengths and weakness of each is covered in [29]. The LiveBuild prototype is designed to interface with Autodesk Revit 2009. Autodesk Revit is a suite of BIM software solutions with unique implementations for each Architects,

Structural Engineers, and Mechanical Engineers. LiveBuild is implemented in the Structural Engineering package, Revit Structures 2009. Autodesk Revit was selected because of its "open" (i.e. freely-available) Application Programming Interface (API) and software development kit (SDK). An API is a collection of methods that allow third-party software developers to write custom software for the parent application. The API and SDK are both available to download on Autodesk's Development Center website [30]. The Revit API allows software to be programmed in Visual Basic, C#, and c++ programming languages [31]. The author chose the C# language for LiveBuild because of prior programming experience with the language. LiveBuild was developed using Microsoft Visual C# 2008 Express Edition to take advantage of its timesaving tools for software development. The LiveBuild prototype uses Microsoft SQL Server database management system, implemented in Microsoft SQL Server 2008 Express. Standard SQL statements are used to post and retrieve data from the database. Both Visual C# 2008 Express Edition and SQL Server 2008 Express are available free of charge on Microsoft's website [32].

3.3.1 Requirements met

Support Sensor Data & Relationship. Within the BIM, the building data is stored in an internal database. As previously mentioned, all BIM software has this database of information which this study refers to as the "implicit database." Revit does not allow direct access to the implicit database through the user interface. In most BIM systems the implicit database may only be modified indirectly by manipulating the building model or by modifying the parameters that are presented in the user interface. Users are never given full access to the database but this is not a drawback for the BIM system. In fact, this is prudent decision made by the Revit developers. Most users do not require access to the database and may disrupt the performance of the building model if given access. Restricted access to the implicit database posed a technical challenge in developing the LiveBuild prototype. The data in the implicit database was not accessible but was needed to extend the database to include sensors. The author overcame the accessibility limits of the implicit database by linking it to an external SQL Server

database. A copy of Revit's implicit database was placed in an external source that was accessibly to LiveBuild (Figure 8).

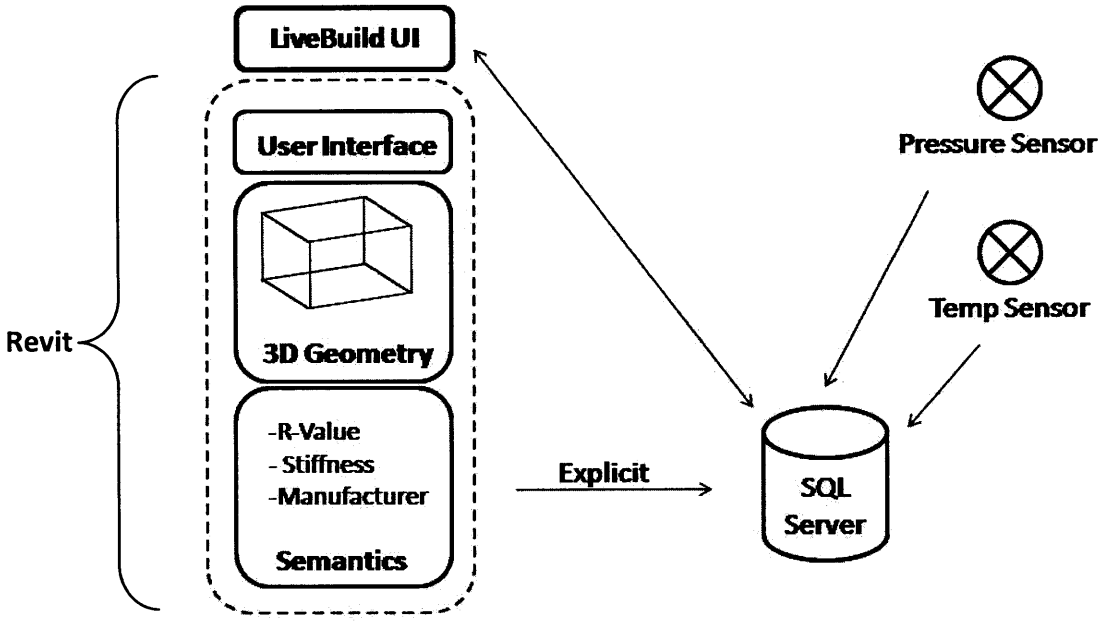

Figure 8: LiveBuild interaction with Revit

Like Revit, the LiveBuild prototype does not fully expose its database to the user. It only allows limited information access through the LiveBuild user interface but keeps system-level information hidden from the user.

Support User/Data Interaction. The LiveBuild user interface (UI) is designed to facilitate easy human interpretation of the information contained in the building model. It supports functions including: relating sensors to building objects, defining constraints on sensor readings and defining the actions taken if the constraint is met. The first prototype, presented in this document, is intended for exploratory use by researchers and design professionals. Though the current user interface is designed to be easily understood by the aforementioned audience, the ultimate goal of this research is to create a system that can be used by building managers and maintenance staff. Future user interfaces will improve accessibility for a broad range of individuals.

Operate in the Native Environment. LiveBuild is implemented within Autodesk Revit 2009 native environment and takes advantage of the Revit user interface. LiveBuild does not require users to learn a new interface. Anyone familiar with the Revit UI can easily operate LiveBuild. Furthermore, LiveBuild does not introduce a new data schema which may perpetuate the problem of interoperability between BIM software. LiveBuild is also the first self-updating building model that operates as an extension to an existing commercial BIM system. This unique feature makes LiveBuild an attractive option for the growing number of professional design firms investing in BIM technology. Stand-alone self-updating modeling software would require additional training which may discourage its use. Therefore, the transition from static to active building models is as simple as installing a LiveBuild plug-in.

Developing real-time functions in the Revit environment present several challenges. The Revit API has a limited number of prescribed methods available to software developers. In some cases, the LiveBuild development worked well within the available API functions. In other cases, the API presented constraints that required the development of an indirect problem solution. For example, the Revit API does not allow the capture of *all* user actions. The API allows the capture of events involving Revit objects (e.g. a user selects a column or beam) but does not allow the capture of high-level events (e.g. mouse clicks or when the "Enter" button is pressed). This limitation presented problems when developing tools that required a back-and-forth interaction between the user and the LiveBuild software. Also, the API has no programmatic link to the underlying Revit database. Consequently, users must manually establish the connection between Revit and the LiveBuild database. This could easily be solved by expanding the API to include a method to access the Revit database. Finally, the API does not allow continuity between software tools. Each function developed through the API is an individual entity. Therefore, variables defined within a tool only exist as long as the tool is active; variables do not persist in memory for future reference. After the function is complete, all local variables are lost. Passing user input data from one function to another required additional effort to handle this complication.

Support Various Building System Control. LiveBuild is designed to control a wide range of systems by allowing generic connection to building systems. The only requirement is that can be programmatically actuated; this allows automatic actuation by LiveBuild. Current state-of-the-art building automation systems have similar requirements for building system. It presents a platform for building optimal package for each building system.

Maintain central persistent storage. LiveBuild uses a central SQL Server database as the single source for persistent storage. Information is never stored locally on client computers and is never stored random access memory (RAM). Each user connects to the database and works from one persistent source. This prevents data loss from a user that is unexpectedly disconnected from the database.

A central data source is used in LiveBuild prototype for simplicity. It does not preclude the use of a distributed network. Even a distributed network may still work as a central database. It would be similar to taking the central database from the LiveBuild and placing portions of the on a different computer. The data would be exactly the same but the distributed network would provide the benefit of using the computational power of several machines instead of just one.

Manage Concurrency. Concurrency conflicts in the model are managed by the Revit environment and concurrency conflicts in the database are managed by Microsoft SQL Server. Revit uses "locking" to manage concurrency in the building model. Once a component within the model is modified by a user it is "locked" to prevent editing by other users. The component is only "unlocked" after the original editor has saved to his/her modifications the central model and has relinquished control of the model component(s).

Revit's "locking" approach seems to have directly grown from the approach of database concurrency management. SQL Server also manages concurrent using a type of "locking." The Transaction Isolation Level (TIL) determines the level of protection from concurrency conflicts. LiveBuild uses a TIL setting of "Read Committed" which prevents

(or locks) information from being read before an operation (transaction) is completed (committed). The challenge is setting proper level of "locking" without unnecessarily reducing the database performance. Increased concurrency protection reduces the performance of the system by increasing latency.

Limit Latency. Latency has not proven to be a problem in LiveBuild prototype. The model performs at latency level that is unnoticeable to a user. This would be more critical in a true "real-time" but LiveBuild is more accurately classified as pseudo-real-time.

3.3.2 Software Functions

The LiveBuild user interface (UI) and the overall model operation are illustrated in Figure 9.

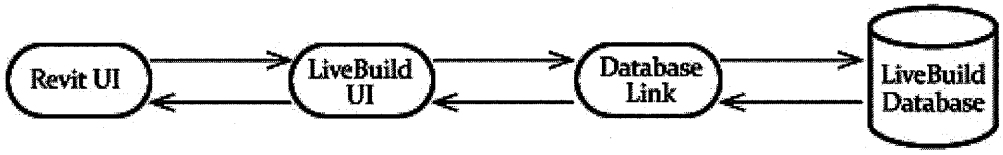

Figure 9: Communication of LiveBuild software components

The two major software components developed for LiveBuild allow communication between the BIM environment—in this case Autodesk Revit—and LiveBuild database. These components, the user interface (UI) and the Database Link, provide all of LiveBuild's functions. The Database Link functions as the data manager while the UI provides the user with access to the data. The following functions are supported either by the Database Link, by the UI, or by both.

3.3.2.1 Maintain Database.

LiveBuild's database is, of course, managed by the Database Link software component. As mentioned in section 3.3.1, (subsection "Operate in native environment") the Revit database must first be manually exported to the LiveBuild database before operation. With the Revit database accessible, the Database Link adds the tables and relationships

shown in Figure 10 to the database. These tables provide the framework to define the relationships between BIM model components (BIM objects) and the data provided by sensors. These virtual relationships mirror the relationship between the physical components and the actual sensors.

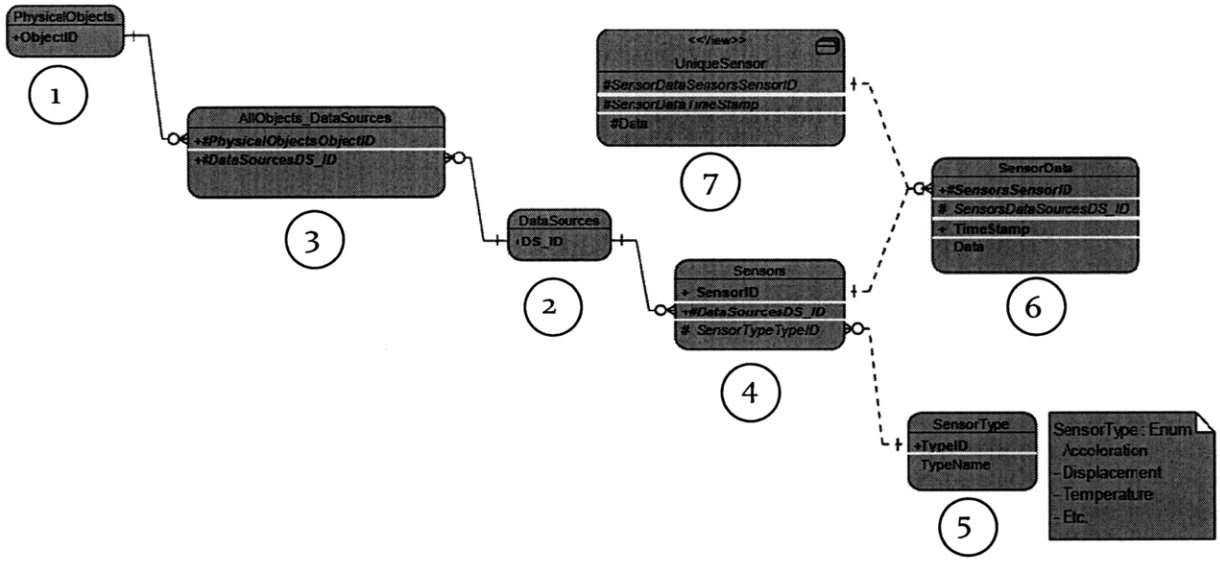

Figure 10: LiveBuild Database Tables Diagram

Figure 10 (above) is a diagram that describes the relationship between the database tables added by the Database Link component. Starting from the left side of the diagram, the tables are as follows:

1) "Physical Objects" contains a list of all objects in the building (and building model)
2) "Data Sources" table contains a list of all the data sources. Data sources may be a single sensor or a collection of sensors in a single node.
3) "All Objects_Data Sources" relates physical objects to data sources.
4) The "Sensors" table lists all the sensors in each data source. As mentioned above, a data source may be made up of several sensors.
5) "Sensor Type" defines all type of information provided by each sensor (acceleration, temperature, pressure, etc.);
6) The "Sensor Data" table holds the data from all the sensors. All sensors post their information to this single table.

7) Each individual sensor has a "View"—or a virtual table extracted from the parent "Sensor Data" table—to represent its data.

Each "View" is extracted from the 'Sensor Data' table by an SQL statement that selects only items with the intended 'Sensor ID' and 'DataSource ID' and then sorts the results by the timestamp. This produces a time history of the data from any Data Source using the "view" but allows a single table to contain the data from every sensor in the system. This method—using a single table for all sensor data with "Views" for each individual sensor—allows simple data posts and read functions. All sensors post data to a single source ('Sensor Data') while all data retrievals come from a table with the same name as the sensor.

3.3.2.2 Manage Database access and BIM Object/Sensor relationships

The UI provides access to the database and to the tables to define relationships. LiveBuild displays a series of windows for user input into database. All windows are launched from within the Revit User Interface (Figure 11). The first window (Figure 12) allows a user to provide the information to connect to the existing database; this information includes the URL where the database is located (for remote access) and login credentials. The next window (Figure 13) shows a list of the data sources (or sensors) that are currently "online." Next, the user is asked to select the Objects in the model to relate to the data sources (Figure 14). Finally, the user is presented a side-by-side list of the data sources and Objects that were selected in the process (Figure 15). This allows a user to make a final selection of the data sources and BIM Objects to relate; their relationship is then stored in the database tables. Figure 16 illustrates the communication between the major software components when a user defines a BIM Object to sensor relationship.

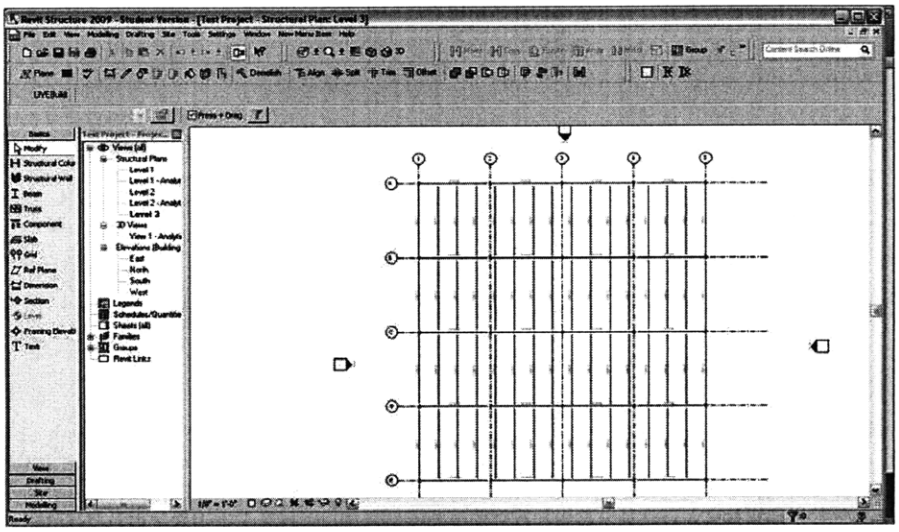

Figure 11: Autodesk Revit User Interface

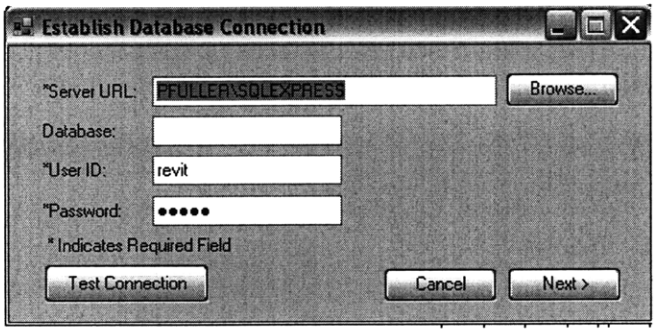

Figure 12: LiveBuild Login

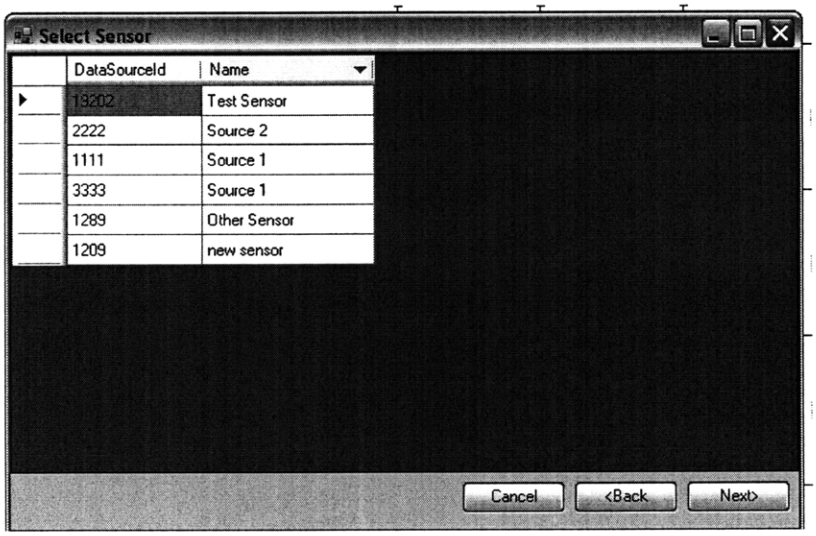

Figure 13: Sensor List

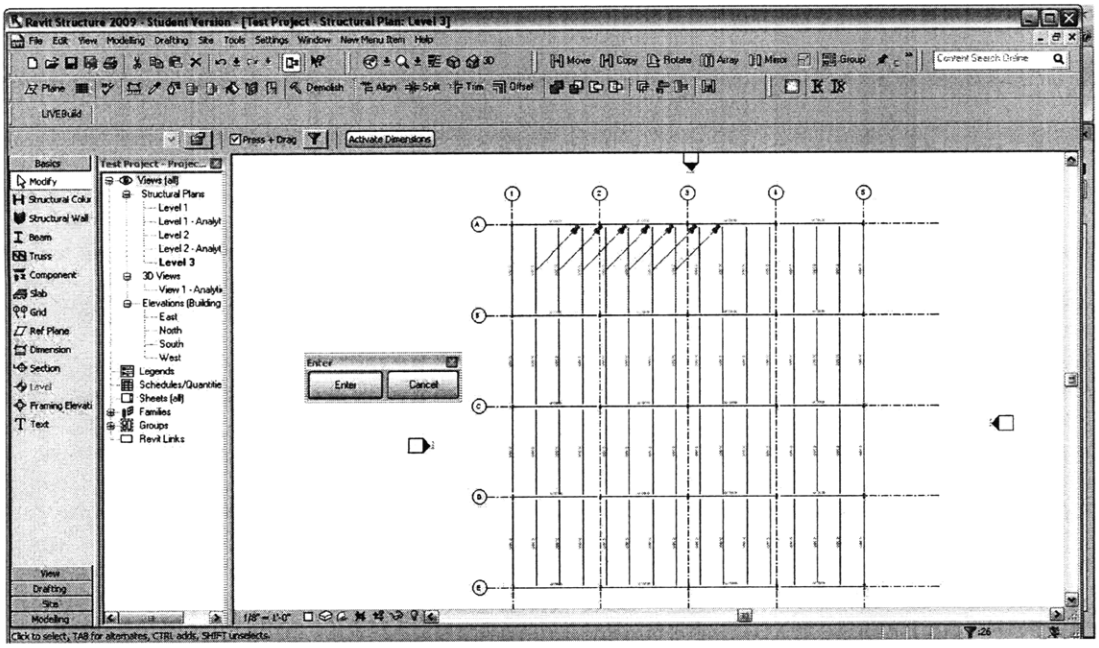

Figure 14: User Selected Objects

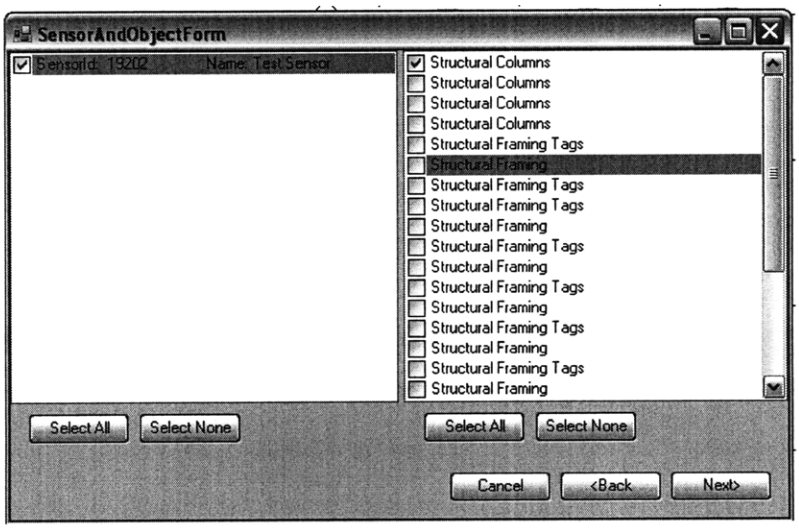

Figure 15: Selects Sensor and Revit Objects

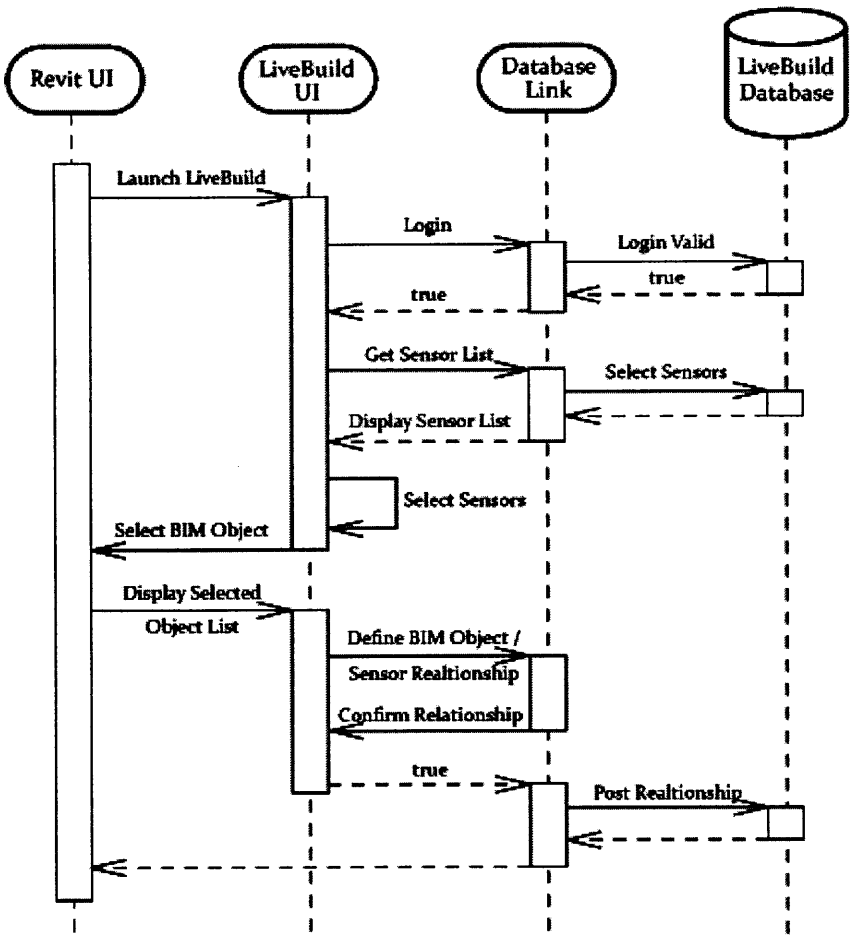

Figure 16: Sequence Diagram: Establish Relationship between BIM Objects and Sensors

It is important to note that these relationships are intended to define the relationship between the sensor data and Objects they describe. They are not intended to define only the sensor's host object. For example, a temperature sensor may be physically attached to a wall (its host) however it may semantically represent the temperature of the entire room. In this case, the database should reflect the relationship between the sensor and the entire room, not just the wall.

3.3.2.3 User-defined event criteria.

The event criterion is managed by both the UI and the Database Link. Event criteria are, for example, constraints placed on the value of the sensor readings (

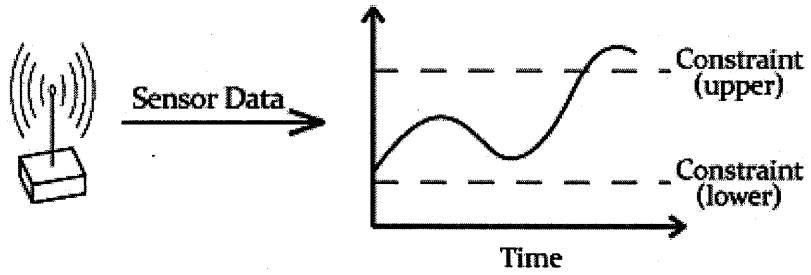

Figure 17). An event occurs when a sensor reading crosses the threshold defined by the criteria.

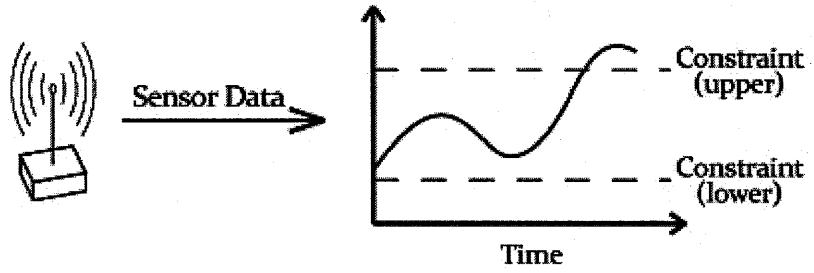

Figure 17: Example LiveBuild Event Criteria

The event criteria must be met to trigger an action (see Section 3.3.2.4). The current LiveBuild prototype includes only simple "greater than" and "less than" criteria (i.e. upper and lower bounds). These "skeleton" functions are hard-coded into LiveBuild but lack a specific numerical value; numerical input from a user is used to define the specific limit. SQL "triggers" are used to implement the criterion. When an event occurs, LiveBuild adds a row to a special table called the Event Table. The Event Table acts as a queue for LiveBuild events. If an event-action relationship exists, the "Actions" are invoked by LiveBuild.

3.3.2.4 Manage LiveBuild "Actions"

Like events, LiveBuild "Actions" are also facilitated by both the UI and Database Link. The "Actions" are literally the action that is initiated when an event occurs. Figure 18 illustrates the communication between the major software components when a user defines event criteria and actions.

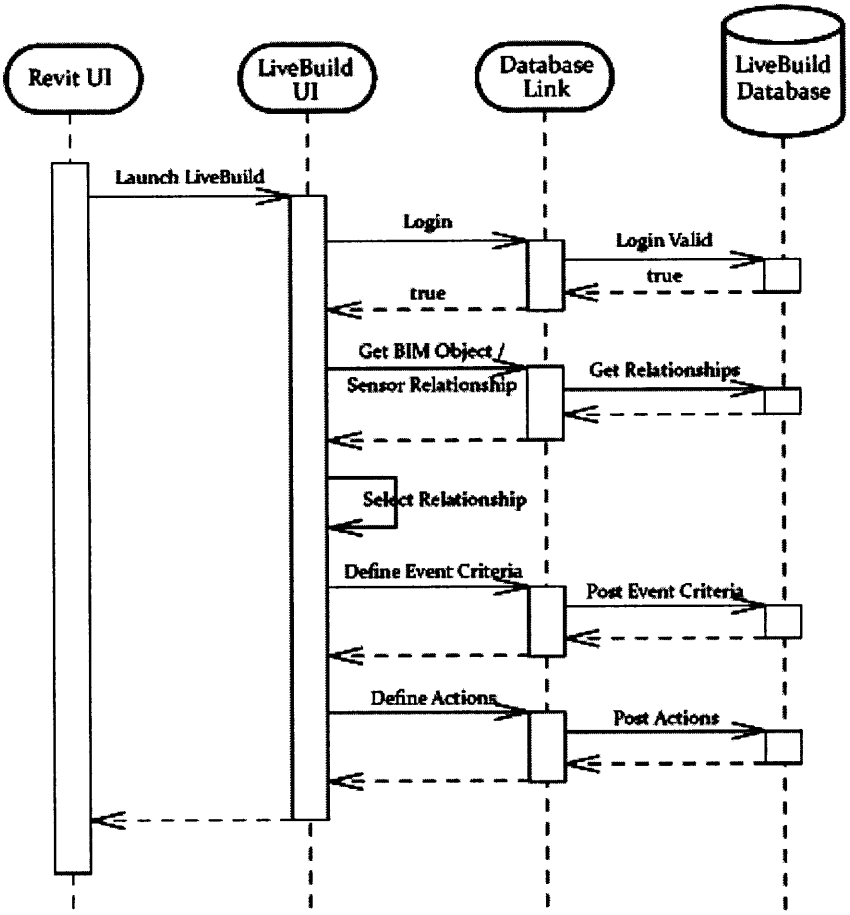

Figure 18: Sequence Diagram: Defining Event Criteria and Actions

Sensors provide LiveBuild with an accurate, up-to-date representation of the building, however "Actions" give the system the power to dynamically respond and modify itself (Figure 18 above). The actions may be simple message/notifications to the user or may be automated modification of building system controls (Figure 19). The default Action in LiveBuild is to send a message to the user. The message may be a pop-up window with information about an event or may appear as a persistent tag near an Object in the building model. The message type used to display the information is dependent upon the nature information being presented. For example, a message regarding a dramatic temperature change in an area of the building may appear as a pop-up message. Conversely, a message regarding a specific sudden drop in pressure in a water pipe may show up as a persistent tag in the building model. The message delivery method suits the scope of the event.

The next and most powerful level of LiveBuild "action" is using the information contained in the BIM to automate building systems control (Figure 19).

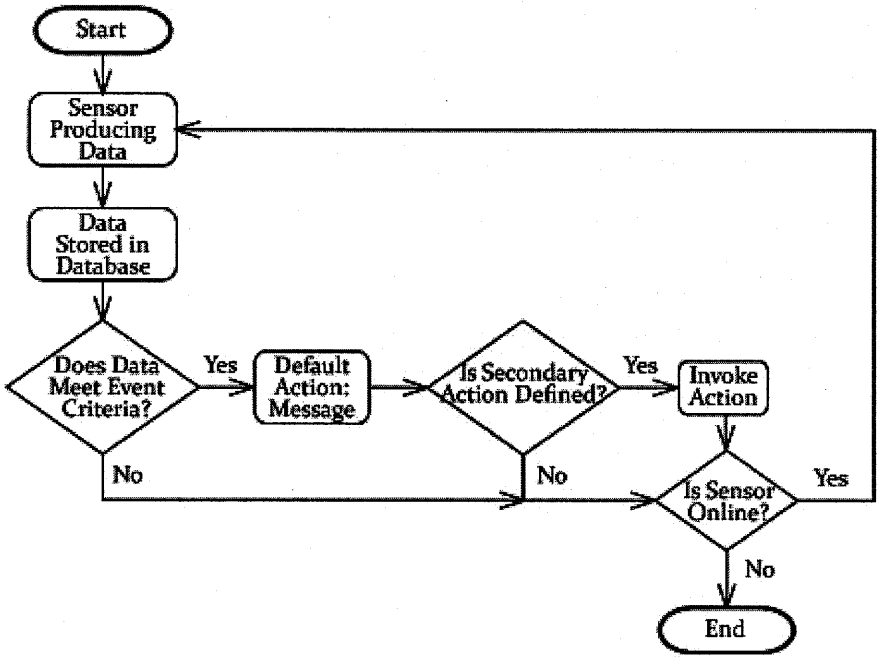

Figure 19: Flowchart: LiveBuild Action

The current implementation of LiveBuild is "wired" with a generic control method—called a "delegate" in the C# programming language—to actuate building systems. It would be a time-consuming task to write a unique method for every possible building system available; the delegate eliminates the need for specific controls methods. Instead, LiveBuild uses a delegate that users may later define with the appropriate action for their unique building system (e.g. turn off light; increase temperature by 1°). *Any* system that can be actuated programmatically (by a machine) can be added as a LiveBuild "action."

3.4 Future Applications

The LiveBuild prototype developed for this investigation is immediately applicable for emergency response systems and construction inspection/verification. LiveBuild information may be used to presented optimal egress routes to occupants in emergency response situations. Likewise, emergency response teams may be given real-time

information regarding a building and an optimal means of entering in an emergency situation.

Engineering firms that currently use BIM software in the design process can employ location sensors on major building components for tracking during the construction process. LiveBuild can track the location of building components on the actual site and virtually present a mirror of the site. This would allow remote construction management and verification of the construction is as designed.

3.5 Other Challenges

This investigation allows off-the-shelf BIM software to respond to real-time data by building upon the existing software platform. The simplicity of LiveBuild's implementation is achieved through the use of an API, however the API also present limitations. The API limits the BIM-client functions accessible for use in LiveBuild and requires a unique mapping for each BIM software that LiveBuild will support.

API functions. An API is a collection of methods that an original software developer makes available for third-party programmers to use in software development. Therefore, a third-developer is limited to use the functions that the original developer allows access to. For example, in Autodesk Revit there are several export functions (to various file formats) that are available in the user interface but are not accessible through the API. Limited access to the BIM-client's functions complicates the development of some desirable LiveBuild functions and makes other functions nearly impossible develop.

Unique implementation. LiveBuild should to be compatible with BIM software from several commercial developers to be useful within the building industry. API's are intrinsically tied to the specific software that they support; LiveBuild's API-based implementation would require a mapping to a unique API for each supported BIM software.

3.6 Future Work

Domain-specific data synthesis. Many research-based BIM/sensor systems are presented as a "general" solution but then use domain-specific test cases for proof-of-concept. Herein lies the problem, a domain-specific proof-of-concept usually involves developing software components for specific use cases that are not generally applicable. However, assuming that a generally acceptable software architecture *does* exists for integrating real-time sensor information, domain-specific problems will still require some additional software development. Sensor-driven systems are not useful unless they are "intelligent" enough to synthesize the large amounts of data into something useful for humans to understand or can automatically take action without human intervention. Future work may include selecting one specific domain and developing systems to synthesize data and present the results in a manner appropriate to that domain.

Latency. Latency is the time it takes for a sensor reading sent to the database to become available within the model. Real-time systems must be able to respond to changes quickly; requiring low latency. LiveBuild has shown that it can respond to changes in the system but its latency has not been determined. Appropriate latency levels can only be determined when a specific implementation is needed; unique systems have unique latency requirements. Future work may include 1) determining the existing latency of LiveBuild and 2) finding ways to maintain a latency level appropriate to a domain-specific implementation.

More complex action/event criteria. The default logical event criteria currently contained in LiveBuild is very basic (Section 3.3.2.3). A more robust system should not only include specific upper and lower bounds but also should be able to interpret user defined equations and recognize patterns in the data. Future work may include more sophisticated criteria to trigger an action.

Accessibility of LiveBuild user interface. The LiveBuild prototype presented in this document is for researched and professional. The ultimate goal is to make it accessible to building owners and maintenance crews. To accomplish this goal, future prototypes

should focus on developing tools and presentation styles that specifically suited for building managers and maintenance crews.

Mobile Device Interface. Modern mobile communication devices are ubiquitous. Future research may explore the opportunity for occupants to "communicate" with a building via mobile devices. This may initially provide support for occupant to "register" their mobile device with the building and allow two-way text-based communication. For example, an occupant may send text-message requests to change the temperature in a specific room. The model may then poll other room occupants and modify the control scheme based on the collective comfort of occupants. Furthermore, emergency response schemes may be communicated to occupants' mobile devices to expedite safe evacuation.

3.7 Conclusions

This paper presents a simplified software architecture for self-updating building information model has been shown to be feasible and advantageous. The experimental software setup presented in this document, using off-the-shelf software components (building information modeling, database) suggest that self-updating models may be improved by simplifying the architecture and implementation. The simplifications include:

- **Reducing the number of custom components.** The LiveBuild prototype uses only off-the-shelf components and can achieve similar results to other more custom systems. This approach keeps the architecture "open" and accessible to many users.
- **Avoiding complicating interoperability.** The addition of custom file types may allow easy data manipulation within a self-updating model but may later complication interoperability with other software. LiveBuild works within the native BIM environment. This may prove to be an advantageous approach for other self-updating building models.

Appendix: LiveBuild Code

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Text;
using Autodesk.Revit;
using Autodesk.Revit.Elements;
using Autodesk.Revit.Enums;

namespace LiveBuild
{
    class Messages
    {
        public Messages() { }

        public static Boolean DialogIsCancelled()
        {
            DialogResult dr = MessageBox.Show("The process is not complete.\n" +
                "Are you sure you want to cancel the LiveBuild Database Connection?",
                "LiveBuild", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (dr == DialogResult.Yes)
                return true;
            else
                return false;
        }

        public void TagSelected(Document activeDoc, ref string messg)
        {
            try
            {
                //Document doc = commandData.Application.ActiveDocument;
                ElementSet sel = activeDoc.Selection.Elements;
                ElementSetIterator se = activeDoc.Selection.Elements.ForwardIterator();
                while (se.MoveNext())
                {
                    Element el = se.Current as Element;
                }
                foreach (Element el in sel)
                {
                    LocationCurve loc = el.Location as LocationCurve;
                    Autodesk.Revit.Geometry.XYZ start = loc.Curve.get_EndPoint(0);
                    Autodesk.Revit.Geometry.XYZ end = loc.Curve.get_EndPoint(1);
                    Autodesk.Revit.Geometry.XYZ mid = start.Add(end).Divide(2);
                    Autodesk.Revit.Elements.View activeView = activeDoc.ActiveView;

                    if (activeView.ViewType != ViewType.ThreeD || activeView.ViewType != ViewType.Undefined)
                    {
                        Autodesk.Revit.Elements.View view = activeView;
                        TagMode tgMode = TagMode.TM_ADDBY_CATEGORY;
```

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Text;
using Autodesk.Revit;

namespace LiveBuild
{

    class DataBaseLink
    {
        private SqlConnection connx;
        private DataSet dataSet;

        public DataBaseLink(){}

        public DataBaseLink(String dbURL, String dbName, string userID, string pass){
            connx = GetConnection(dbURL, dbName, userID, pass);
        }

        public DataBaseLink(String connectionString)
        {
            connx = GetConnection(connectionString);
        }

        public Boolean IsValidConnection()
        {
            if (connx.ConnectionString == null)
            {
                return false;
            }
            else
            {
                try
                {
                    connx.Open();
                    if (connx.State.ToString() == "Open")
                    {
                        connx.Close();
                        return true;
                    }
                    else
                    {
                        connx.Close();
                        return false;
```

```
                }
            }
            catch (Exception ex)
            {
                DataBaseLink.DBLinkErrorMessage(ex);
            }
            return false;
        }
    }

    public void BuildDataSet(String newDataSetTableName, String selectedColumns, String fromTables)
    {
        //call static method
        dataSet = DataBaseLink.GetDataSet(this.connx, newDataSetTableName, selectedColumns, fromTables);
    }

    public DataSet DataSet
    {
        get { return dataSet; }
    }

    public SqlConnection Connection
    {
        get { return connx; }
        set { connx = value; }
    }

    public static string GetConnectionString(String dbURL, String dbName, string userID, string pass)
    {
        SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
        builder.DataSource = dbURL;
        builder.InitialCatalog = dbName;
        builder.UserID = userID;
        builder.Password = pass;
        builder.ConnectTimeout = 20; //time in seconds
        return builder.ConnectionString;
    }


    public static SqlConnection GetConnection(String dbURL, String dbName, string userID, string pass)
    {
        String connectionString = GetConnectionString(dbURL, dbName, userID, pass);
        SqlConnection con = new SqlConnection(connectionString);
        return con;
    }

    public static SqlConnection GetConnection(String connectionString)
    {
```

```
            SqlConnection con = new SqlConnection(connectionString);
            return con;
        }

        public static void TestSQLConnection(SqlConnection connection)
        {
            try
            {
                connection.Open();
                if (connection.State.ToString() == "Open")
                {
                    MessageBox.Show("Connection Successful", "SQL Connection State", MessageBoxButtons.OK, MessageBoxIcon.
    Information);
                }
                else
                {
                    MessageBox.Show("Connection Failed\nConnection state:\t" + connection.State.ToString() +
                                    "Please try again", "SQL Connection State", MessageBoxButtons.OK, MessageBoxIcon.
    Error);
                }
                connection.Close();
            }
            catch (Exception ex)
            {
                DataBaseLink.DBLinkErrorMessage(ex);
            }
        }

        public static void TestSQLConnection(String dbURL, String dbName, string userID, string pass)
        {
            TestSQLConnection( GetConnection(dbURL, dbName, userID, pass) );
        }

        public static DataSet GetDataSet(SqlConnection connection, String newDataSetTableName, String
    selectDataBaseColumns, String fromDataBaseTables)
        {

            //create adapter; will eventually use it to fill DataSet
            SqlDataAdapter sensorTableAdapter = new SqlDataAdapter();

            //Create a DataSet Table and name it
            sensorTableAdapter.TableMappings.Add("Table", newDataSetTableName);

            //create a query string to selects rows from database
            String query = SelectStatementString(selectDataBaseColumns, fromDataBaseTables);

            DataSet dSet = new DataSet();
            try
```

```
            {
                //connection must be open when it is passed into the SqlCommend
                connection.Open();

                //create the SqlCommand object from query string and connection
                SqlCommand command = new SqlCommand(query, connection);
                command.CommandType = CommandType.Text;

                sensorTableAdapter.SelectCommand = command;

                //can create another adapter here to add other tables to the Dataset
                sensorTableAdapter.Fill(dSet);
                //close connection
                connection.Close();
            }
            catch (Exception ex)
            {
                DataBaseLink.DBLinkErrorMessage(ex);
            }
            return dSet;
        }


        public static void DBLinkErrorMessage(Exception ex)
        {
            SqlException sqlEx = ex as SqlException;
            if (sqlEx != null)
            {
                String errorMessages = "";
                for (int i = 0; i < sqlEx.Errors.Count; i++)
                {
                    errorMessages += "Index #" + i + "\n" +
                        "Message: " + sqlEx.Errors[i].Message + "\n" +
                        "LineNumber: " + sqlEx.Errors[i].LineNumber + "\n" +
                        "Source: " + sqlEx.Errors[i].Source + "\n" +
                        "Procedure: " + sqlEx.Errors[i].Procedure + "\n";
                }
                MessageBox.Show(errorMessages + "\n" + ex.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error) ↙
;
            }
            else
                MessageBox.Show(ex.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        public static String SelectStatementString(String columnNames, String tableNames)
        {
            String s = "SELECT " + columnNames + " FROM " + tableNames;
            return s;
```

```
        }

        public static void GetObjectData(int ObjectId, SqlConnection connx){

            String dataString="";
            //get list of data sources associated with this object
            int[] dataSources = DataBaseLink.GetDataSources(ObjectId, connx);
            if (dataSources == null || dataSources.Length == 0)
            {
                MessageBox.Show("No sensor data available for Object " + ObjectId, "Error", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
            }
            else
            {
                //for each data source, find all sensors
                for (int i = 0; i < dataSources.Length; i++)
                {
                    dataString += "Data Source ID: " + dataSources[i] + "\n";
                    //get list of sensors
                    int[] sensors = DataBaseLink.GetSensors(dataSources[i], connx);
                    if (sensors == null || sensors.Length == 0)
                        dataString += "(No Sensors Online)";
                    else
                    {
                        //for each sensor, get current data
                        for (int j = 0; j < sensors.Length; j++)
                        {
                            dataString += "\tSensor ID: " + sensors[j] + ", ";
                            dataString += DataBaseLink.GetCurrentSensorDataAsString(dataSources[i], sensors[j], connx) +
    "\n";
                        }
                    }
                }
                MessageBox.Show(dataString, "Current Sensor Information", MessageBoxButtons.OK, MessageBoxIcon.
    Information);
            }
        }

        public static int[] GetDataSources(int ObjectId, SqlConnection connx)
        {
            //create adapter; will eventually use it to fill DataSet
            SqlDataAdapter sensorTableAdapter = new SqlDataAdapter();

            //Create a DataSet Table and name it
            sensorTableAdapter.TableMappings.Add("AllObjects_DataSource", "DataSource_Sensors");

            //create a query string to selects rows from database
            String query = "SELECT DataSourceId FROM AllObjects_DataSource WHERE ObjectId=" + ObjectId;
```

```
        DataSet dSet = new DataSet();

        //connection must be open when it is passed into the SqlCommend
        connx.Open();

        //create the SqlCommand object from query string and connection
        SqlCommand command = new SqlCommand(query, connx);
        command.CommandType = CommandType.Text;

        sensorTableAdapter.SelectCommand = command;

        //can create another adapter here to add other tables to the Dataset
        sensorTableAdapter.Fill(dSet);
        //close connection
        connx.Close();

        int count = dSet.Tables[0].Rows.Count;
        int[] dataSourceList = new int[count];
        for (int i = 0; i < count; i++)
        {
            String dataSourceIdAsString = dSet.Tables[0].Rows[i][0].ToString();
            dataSourceList[i] = int.Parse(dataSourceIdAsString);
        }
        return dataSourceList;
    }

    public static int[] GetSensors(int dataSourceId, SqlConnection connx)
    {
        //create adapter; will eventually use it to fill DataSet
        SqlDataAdapter sensorTableAdapter = new SqlDataAdapter();

        //Create a DataSet Table and name it
        sensorTableAdapter.TableMappings.Add("Sensors", "SensorList");

        //create a query string to selects rows from database
        String query = "SELECT DISTINCT SensorId FROM Sensor WHERE DataSourceId=" + dataSourceId;

        DataSet dSet = new DataSet();

        //connection must be open when it is passed into the SqlCommend
        connx.Open();

        //create the SqlCommand object from query string and connection
        SqlCommand command = new SqlCommand(query, connx);
        command.CommandType = CommandType.Text;

        sensorTableAdapter.SelectCommand = command;
```

```
        //can create another adapter here to add other tables to the Dataset
        sensorTableAdapter.Fill(dSet);
        //close connection
        connx.Close();

        int count = dSet.Tables[0].Rows.Count;
        int[] sensorList = new int[count];
        for (int i = 0; i < count; i++)
        {
            String sensorIdAsString = dSet.Tables[0].Rows[i][0].ToString();
            sensorList[i] = int.Parse(sensorIdAsString);
        }
        return sensorList;
    }

    public static String GetCurrentSensorDataAsString(int dataSourceId, int sensorId, SqlConnection connx)
    {
        String sensorData="";
        //create adapter; will eventually use it to fill DataSet
        SqlDataAdapter sensorTableAdapter = new SqlDataAdapter();

        //Create a DataSet Table and name it
        sensorTableAdapter.TableMappings.Add("Table", "SensorData");

        //create a query string to selects rows from view in the database
        String query = "SELECT TOP 1 * FROM ["+ dataSourceId + "_" + sensorId + "] ORDER BY TimeStamp DESC";

        DataSet dSet = new DataSet();

        //connection must be open when it is passed into the SqlCommend
        connx.Open();

        //create the SqlCommand object from query string and connection
        SqlCommand command = new SqlCommand(query, connx);
        command.CommandType = CommandType.Text;

        sensorTableAdapter.SelectCommand = command;

        //can create another adapter here to add other tables to the Dataset
        sensorTableAdapter.Fill(dSet);
        //close connection
        connx.Close();

        //column[0] = timestamp
        //column[1] = sensor reading/data
        //column[2] = sensor type (acceleration, temprature, etc)
```

```
            int count = dSet.Tables["SensorData"].Rows.Count;
            int[] sensorList = new int[count];
            if (count == 0)
                return "(No current data available)";
            else
            {
                for (int i = 0; i < count; i++)
                {
                    String s = dSet.Tables["SensorData"].Rows[i][2].ToString();
                    s = s.Trim();
                    sensorData += s + " Value = " + dSet.Tables["SensorData"].Rows[i][1].ToString() +
                        " ( at time = " + dSet.Tables["SensorData"].Rows[i][0].ToString() + ")";
                }
            }
            return sensorData;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Windows.Forms;
using Autodesk.Revit;
using Autodesk.Revit.Elements;
using Autodesk.Revit.Structural.Enums;

namespace LiveBuild
{
    class LBApplication : IExternalCommand
    {
        DBConnxForm connectionForm;
        public LBApplication() { }

        public LBApplication(ExternalCommandData comdData, ref string mesg, ElementSet elems)
        {
            try
            {
                Document activeDoc = comdData.Application.ActiveDocument;
                connectionForm = new DBConnxForm(activeDoc);
                connectionForm.Show();
            }
            catch (Exception ex)
            {
                DataBaseLink.DBLinkErrorMessage(ex);
            }
        }


        public IExternalCommand.Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
        {
            LBApplication app = new LBApplication(commandData, ref message, elements);
            return IExternalCommand.Result.Succeeded;
        }


    }
}
```

```
                    TagOrientation tgOrientation = TagOrientation.TAG_HORIZONTAL;
                    IndependentTag newTag = activeDoc.Create.NewTag(view, el, true, tgMode, tgOrientation, mid);
                }
            }
        }
        catch (Exception e)
        {
            messg = "Exception thrown in LiveBuild.TestCommand: \n" + "Source:\t\t" + e.Source + "\n\nCalling Method:\t" + e.TargetSite.ToString() + "\n\nStack Trace:\n" + e.ToString() + "\n\nHelp:\t\t" + e.HelpLink;
        }
    }

    /// <summary>
    /// This method iterates over a ParameterSet and builds a list (string)
    /// of all the Parameters and their values in the following form
    /// ParameterName: ParameterValue
    /// </summary>
    /// <param name="pSet"></param>
    /// <param name="dc"></param>
    /// <returns></returns>
    public static string GetParamListAsString(Element elem, Document dc)
    {
        ParameterSet pSet= elem.Parameters;
        System.Text.StringBuilder strng = new System.Text.StringBuilder();
        foreach (Parameter p in pSet)
        {

            String defName = p.Definition.Name;
            switch (p.StorageType)
            {
                case Autodesk.Revit.Parameters.StorageType.Double:
                    strng.Append(defName + " : " + p.AsDouble() + " (Double)\t");
                    break;
                case Autodesk.Revit.Parameters.StorageType.ElementId:
                    ElementId id = p.AsElementId();
                    if (id.Value >= 0)
                    {
                        strng.Append(defName + " : " + dc.get_Element(ref id).Name + " (Element ID)\t");
                    }
                    else
                    {
                        strng.Append(defName + " : " + id.Value.ToString() + " (Element ID)\t");
                    }
                    break;
                case Autodesk.Revit.Parameters.StorageType.Integer:
                    if (p.Definition.ParameterType == Autodesk.Revit.Parameters.ParameterType.YesNo)
                    {
                        if (p.AsInteger() == 0)
```

```
                            {
                                strng.Append(defName + " : False (Integer-Yes/No)\t");
                            }
                            else
                                strng.Append(defName + " : True (Integer-Yes/No)\t");
                        }
                        else
                        {
                            strng.Append(defName + " : " + p.AsInteger() + " (Integer)\t");
                        }
                        break;
                    case Autodesk.Revit.Parameters.StorageType.String:
                        strng.Append(defName + " : " + p.AsString() + " (String)\t");
                        break;
                    default:
                        strng.Append(defName + " : Undefined Parameter\t");
                        break;
                }

            }
            return strng.ToString();
        }
    }

}
```

REFERENCES

[1] *US National Building Information Modeling Standard,* Washington D.C.: National Institute of Building Sciences (NIBS), 2007.

[2] A. Mahdavi, "Aspects of Self-aware Building," *International Journal of Design Sciences and Technology,* vol. 9, 2001, pp. 35-52.

[3] K.A. Brunner and A. Mahdavi, "A Software Architecture for Self-updating Life-cycle Building Models," *Computer Aided Architectural Design Futures 2005,* 2005, pp. 423-432.

[4] K.A. Brunner and A. Mahdavi, "The software design of a dynamic building model service," *Proceedings of the 22nd CIB W,* 2005.

[5] "Phone Interview with Thornton-Tomasetti Associate," Sep. 2008.

[6] "GSA - Spatial Program Validation," *US General Services Administration,* Jul. 2008.

[7] M. Blackwell, "Texas Adopts Building Information Modeling (BIM) capability for State Design and Construction Projects — Texas Facilities Commission," *Texas Facilities Commission.*

[8] "BIM Implementation Announcement," Jun. 2009.

[9] "VA Adoption of Building Information Modeling (BIM)," Apr. 2008.

[10] B. Gilligan and J. Kunz, *VDC Use in 2007: Significant Value, Dramatic Growth, and Apparent Business Opportunity,* Stanford University, Center for Integrated Facility Engineering, 2007.

[11] "Whole Building Design Guide," *National Institue of Building Sciences.*

[12] "USGBC: Intro - What LEED Is," *US Green Building Council.*

[13] R.W. Liebing, *Construction of Architecture,* John Wiley and Sons, 2008.

[14] Associated General Contractors of America., *Project delivery systems for construction.,* [Washington D.C.]: Associated General Contractors of America, 2004.

[15] W. Kymmell, *Building information modeling,* McGraw-Hill Professional, 2008.

[16] C.N. Eastman, *Spatial synthesis in computer-aided building design,* Elsevier Science Inc. New York, NY, USA, 1975.

[17] C. Eastman, "The Use of Computers Instead of Drawings," *AIA Journal,* vol. 63, 1975, pp. 46–50.

[18] K.H. Law and M.K. Jouaneh, "Data Modeling for Building Design," *Proceedings of the 4th Computing Conference in Civil Engineering*, 1986.

[19] W.F. Gielingh, *General reference model for AEC product definition data*, Delft, The Netherlands: TNO - IBBC, 1987.

[20] B.C. Björk, "Basic structure of a proposed building product model," *Computer Aided Design*, vol. 21, 1989, pp. 71-78.

[21] "Industry Foundation Classes," *buildingSMART International*, Jan. 2009.

[22] J. Song, C.T. Haas, and C.H. Caldas, "Tracking the Location of Materials on Construction Job Sites," *Journal of Construction Engineering and Management*, vol. 132, 2006, p. 911.

[23] F. Caron, S.N. Razavi, J. Song, P. Vanheeghe, E. Duflos, C. Caldas, and C. Haas, "Locating sensor nodes on construction projects," *Autonomous Robots*, vol. 22, 2007, pp. 255-263.

[24] A.M.R. Ward, "Sensor Driven Computing," Ph.D. Thesis, University of Cambridge, 1998.

[25] M. Hazas, J. Scott, and J. Krumm, "Location-aware computing comes of age," *Computer*, vol. 37, 2004, pp. 95-97.

[26] O. Icoglu, K.A. Brunner, A. Mahdavi, and G. Suter, "A Distributed Location Sensing Platform for Dynamic Building Models," *Ambient Intelligence*, 2004, pp. 124-135.

[27] D.L.D. Ipiña, P.R.S. Mendonca, and A. Hopper, "TRIP: a Low-Cost Vision-Based Location System for Ubiquitious Computing," *Personal and Ubiquitous Computing Journal*, vol. 6, 2002, pp. 206 - 219.

[28] S. Sharples, V. Callaghan, and G. Clarke, "A multi-agent architecture for intelligent building sensing and control," *Sensor Review*, vol. 19, 1999, pp. 135 - 140.

[29] C.M. Eastman, P. Teicholz, R. Sacks, and K. Liston, *BIM Handbook*, John Wiley and Sons, 2008.

[30] "Autodesk - Developer Center ," *http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=2484975*, 2009.

[31] "Revit 2009 API Developer's Guide," Sep. 2008.

[32] "Microsoft Corporation," *http://www.microsoft.com/express/download/*.