BIMSL: A Generic Approach to the Integration of Building Information Models with Real-time Sensor Data

Abstract
The surge of interest in digital building models combined with increasing sensorization of spaces is prompting an integration of Building Information Modelling (BIM) with real-time sensor data. However, current approaches reported in literature either remain theoretical or target very specific application domains, showing that there is no generic approach to assist in the creation of service and application software that combines sensor data with BIM models.

The solution presented in this article addresses the engineering complexity associated with integrating sensor data with BIM by leveraging an advanced software engineering technique known as a Domain-Specific Language (DSL). We demonstrate also that the language herein proposed, named Building Information Modeling Sensor Language (BIMSL) provides substantial gains of expressiveness and ease of use in developing queries that process sensor data with complex conditions over BIM models. BIMSL is validated with experienced software developers according to a consistent evaluation protocol for DSLs focused on effectiveness, efficiency, satisfaction, and usability attributes. The results outperform the standard existing alternatives, indicating that our proposal contributes to reducing the human effort associated with integrating BIM with sensor data.

Keywords: Building Information Modeling, BIM Sensor Language, Internet of Things, Sensors, Real-Time Data, Domain-Specific Language

1. Introduction

The evolution of the Internet of Things is leading to an increase in sensorization of physical spaces [1], resulting in a growing number of applications that require reasoning about their surrounding environment in real-time [2]. These applications benefit greatly from integration with BIM models [1], which makes it possible to digitally represent physical and functional characteristics of physical spaces [3, 4, 5] and may, thereby, provide relevant information about the buildings.

The above-mentioned integration simplifies the engineering process underlying the combination of sensor data with the physical elements and constructive characteristics of the building (required, for instance, to compare the measured performance with the expected performance of the designed building). Conceivably, an integrated approach to handle BIM models and sensor data will bring about a more structured development practice as the engineer conceiving the system has a more precise understanding of the entire data framework.

Despite being highly regarded by the scientific community [6], BIM does not yet have strong capabilities to process real-time data [5], as do other solutions for sensors in smart environments. A number of challenges have to be overcome in order to integrate real-time data with BIM models. One of the main challenges is to retrieve specific information out of BIM [7, 8], especially by using the Industry Foundation Class (IFC) product model [9], which is the most widely recognized model format to capture information of BIM models [8]. Another challenge has to do with the fact that real-time sensor data must be continuously processed, which involves complex processing techniques [10, 11, 12]. As we will make clear later (Section 2), overall, combining BIM and real-time data processing is a complex undertaking.

Several approaches have been proposed to solve above-mentioned challenges of integrating BIM with real-time data [13, 14, 15, 5]. However, most of them lack practical validation [14, 16] or are highly dependent on domain specifications [17]. In fact, several approaches are related to the development of applications concerning specific application domains, namely (i) Energy Management (EM) [18, 19, 20, 5, 21], (ii) Building Automation (BA) [22, 23, 24], (iii) Fire Control [25], (iv) Health and Safety [15], (v) Safety Risk [26], and (vi) Augmented Reality [27]. None of these is adapt-
able to a wide range of application domains.

The development of applications that combine BIM with real-time sensor data, similarly to other data-intensive applications, is hampered by the difficulty to specify correct queries that explore the data sources and feed data into the application. In our particular setting, the queries combine sensor data with complex characteristics of the built environment. Thus, our generic solution to the problem is one that (i) takes into account an updated building model, (ii) is able to integrate data from multiple sensors, (iii) resolves queries that combine sensor with building data, (iv) produces answers for those queries in real-time in terms of the sensor data processed, and (v) offers a clean, easy-to-use, and simple interface to the user (developer) lowering the overall effort associated to application development.

Aiming at the development of a standard solution to support BIM and sensor data integration, this article proposes a Domain-Specific query language for performing real-time queries over sensor data while embedding information from IFC-encoded BIM models. In particular, we describe (i) the BIMSL domain specific language, (ii) its target execution platform, (iii) a mechanism integrated in this target platform that deals with sensor data processing, and (iv) a mechanism that enables querying a BIM model. Domain Specific Languages are an advanced software engineering technique that is known for drastically simplifying the development applications in complex domains (see Section 2.4). This is achieved by providing software developers with limited knowledge of BIM and real-time data processing the adequate abstractions in the form of a specialized language.

In order to demonstrate that the proposed solution is adaptable and suitable to a diversity of situations, a case study implementing several literature-based queries requiring the integration between BIM and sensor data is performed. Furthermore, BIMSL is validated according to a state-of-the-art DSL usability evaluation protocol recommended by the scientific community to assess the effectiveness, efficiency, satisfaction, and accessibility of the user when using BIMSL against an alternative General Purpose Language (GPL) solution.

The next section presents the challenges associated with combining BIM with sensor data processing. The section also details the review of literature that was performed to identify the gaps in the research literature that led to the research described in this article. The language development methodology is presented in Section 3, and Section 4 details BIMSL and its underlying execution framework. Sections 5 and 6 describe the evaluation process and results, respectively.

2. Background

2.1. BIM and IFC

BIM models are digital representations of real-world elements and characteristics of a building. These are described by BIM schemas. Schemas are standardized data structures that allow distinct tools to create and interpret a BIM model [28]. Exchange formats, such as IFC [5], use specific schemas to facilitate the flow of information between applications.

IFC is conceived to capture physical aspects about Building Management Systems (BMSs) through the IfcBuildingControlsDomain schema, which contains all the classes related to automation nodes, such as sensors and actuators. However, the constructs related to this schema allow only the specification of abstract types and textual descriptions, lacking specific information about how they are commissioned.

IFC was not designed to handle time-variant (i.e., real-time and historical) data and, as we make clear in this article, it does not need to. The DSL proposed herein creates an abstraction that gives the engineer the idea that IFC is extended with sensor data. Therefore, queries expressed in the DSL reconcile IFC and sensor data, regardless of the fact that these data come from different sources.

2.2. Real-Time Data Applications to BIM

Several approaches regarding the integration of BIM and sensor data have been proposed in the literature [14, 23, 29, 18, 26, 16, 15]. However, the potential of BIM and sensors integration has not been fully realized as most approaches remain theoretical, i.e., do not have a proper implementation, and are highly dependent on a specific domain, and thus not easily generalizable. A standard solution that simplifies the creation of applications requiring this integration is still missing. Most approaches have been implemented using GPLs, while many of the remaining proposals have not even been implemented yet. Therefore, the existing approaches to this subject are still performed in an adhoc way using GPLs [14, 29, 18]. As a result, the integration process is considerably complex, especially for software developers who do not have experience in BIM and real-time data processing.

2.3. BIM Querying Approaches

In order to deal with this large amount of data, several approaches on BIM querying have been advanced by the scientific community. The most important approaches include the Express Query Language [30], the Partial
Model Query Language [31], the Generalized Model Subset Definition schema [32], the Georgia Tech Process to Product Modeling [33], the DSL Building Environment Rules and Analysis [34], the mvdXML format [35], the Solibri Model Checker application, and the DSL Building Information Model Query Language (BIMQL) [8]. However, none of the above mentioned approaches considers real-time data, which may be related to highly dependent specifications associated with a specific application domain.

Mazairac and Beetz [7, 8] proposed a DSL named BIMQL, which, at the time of writing, is the most important approach to BIM querying, as it seeks to be a standard solution to query BIM models. Nevertheless, there are several aspects that are missing in this solution in order to process real-time data. From a technical point of view, the BIMQL platform lacks a Data Stream Management System (DSMS) as well as spatial query operators. A DSMS [11] is a Stream Processing Engine (SPE) [12] that is able to process several input data streams from a wide diversity of sources and produce a new output stream as a result [36]. DSMSs are regarded as an evolution of regular database systems in order to support the processing of data streams.

2.4. Domain-Specific Languages

Domain-Specific Languages are specialized languages that provide specific constructs and vocabulary to model a problem explicitly (or, in certain cases, a solution for a problem). Conceivably, these benefits extend to construction as well. DSLs have been used in many application domains including automotive, aerospace, and telecom industries, and have well-documented benefits [37]. While GPLs are still most commonly used, DSLs are a state-of-the-art Software Engineering technique that has been gaining momentum in creating solutions for complex problems. With appropriate and simple (but not simpler) descriptions of the intended system, DSLs enable complexity to be virtually reduced to a manageable level, thereby enhancing the productivity of domain users (typically developers themselves). Comparing DSLs with GPLs, two distinctive aspects are important:

Solution synthesis. A GPL is conceived to describe one (and only one) computational solution to the problem (typically manually). In contrast, on a DSL the problem is expressed as a model that is used to automatically synthesize a (computational) solution for the problem. This results in less human effort to create the computational solution.

Semantic gap. By definition a DSL enables specifying (modelling) reality using high-level, domain-specific constructs. In contrast, a GPL forces the user of the language to think in terms of low-level solution constructs. The cognitive distance of the former scenario is recognizably lower than the latter. Therefore, DSLs enable closing the semantic gap between the problem and the actual solution.

Therefore, as evidenced, a standard solution to support the development of applications requiring the integration of BIM and sensor data should be based on DSL, which could be easily adapted to different application domains.

3. Methodology

3.1. Language Design and Development

The DSL solution proposed herein follows a consistent DSL development methodology proposed in literature. The iterative process proposed by Barišić et al. consists of five stages, namely (i) Domain Analysis [38], (ii) Language Design [38], (iii) Language Implementation [38], (iv) Language Deployment [39], and (v) Language Evaluation [38]. These stages contribute to the formal definition of the DSL life cycle [39], which is summarized in Figure 1.

The definition of the DSL implied the development of the target platform that implements the emulator of the language semantics on which queries expressed in BIMSL language can be installed. We call this platform the BIMSL Application Programming Interface (API). The BIMSL API has the purpose of (i) taking into account an updated building model, (ii) integrating data from multiple sensors, (iii) resolving queries that combine sensor data with building data, and (iv) producing answers for those real-time queries in terms of processed data. After developing both the BIMSL language and the BIMSL API, several BIMSL queries are implemented and validated.

3.2. Validation

In order to further validate BIMSL, several hypotheses regarding BIMSL usability were stated and tested using the following methods: (i) interpreting graphics and tables, (ii) comparing the performance of informed participants against uninformed participants, and (iii) performing Wilcoxon Signed-Rank Tests. The Wilcoxon Signed-Rank Test is a non-parametric statistical test that analyses matched-pair data [40, 41], and is used to compare two sets of scores coming from the
same participants [40, 41]. This technique does not assume normality in data and enables testing the hypothesis that the probability distribution of the first sample is equal to the probability distribution of the second sample [40, 41]. It is normally used when one wishes to investigate any change in scores from one point to another. Thus, we will use it to assess whether the differences between BIMSL and Java GPL implementation are statistically meaningful in the context of our problem. In this context, two types of participants are considered:

Informed participants. These are software developers who have previous experience in BIM and/or real-time data processing, and are regular users of GPLs.

Uninformed participants. These are software developers who do not have previous experience in BIM and/or real-time data processing, and are also regular users of GPLs.

The hypotheses tested during the Evaluation Session were:

- **H1null.** Using BIMSL or Java has no impact on the user’s effectiveness in specifying queries that integrate BIM with sensor data.
- **H1alt.** By using BIMSL, a user can specify queries that integrate BIM with sensor data more effectively than when using Java.
- **H2null.** Using BIMSL or Java has no impact on the user’s efficiency in specifying queries that integrate BIM with sensor data.
- **H2alt.** By using BIMSL, a user can specify queries that integrate BIM with sensor data more efficiently than when using Java.
- **H3null.** Using BIMSL or Java has no impact on the user’s confidence in specifying queries that integrate BIM with sensor data.
- **H3alt.** By using BIMSL, a user can specify queries that integrate BIM with sensor data with more confidence than when using Java.

4. BIMSL Development

4.1. Domain Analysis

The first stage of DSL development defines the core language model of the DSL [42]. This stage elicits the
essential domain concepts and knowledge that will inform the language design [43] by taking into account certain domain terms, expressions, and constraints intrinsic to the problem solution [39]. Developing a vocabulary that is easy for domain experts to use is very important [44].

The Domain Analysis was boot-strapped with the collaboration of a domain expert in order to identify concepts, terms, and expressions relevant to the problem. This domain expert has a very solid background in real-time data processing, including several years of experience in this field of activity and knowledge of BIM models. In addition, with a literature-based survey of queries combining BIM with sensor data, it was possible to trace and identify the required characteristics that the DSL must provide. By means of the literature-based survey of queries combining BIM and sensor data, a list of domain concepts, a feature model, and a domain model relating each of the concepts, it was established that, coarsely speaking, the BIMSL queries should, above all:

a) Determine the position of an IFC object.
b) Select attributes or properties from IFC objects.
c) Establish containment relationships between IFC objects.
d) Calculate the distance between two IFC objects.
e) Process real-time data that is provided by sensors (the query received by the system must be transformed into an event processing query that will deal with real-time data processing).
f) Filter sensor measurements, using time window bounds.
g) Apply aggregation functions on a result set.
h) Build a table with attributes, properties, and measurements bounded by a specified window as columns. Cartesian Product may be applied.
i) Apply filters to records and extract only the records that fulfill specified criteria.
j) Group a result set by one or more of its columns.
k) Apply filters over aggregated results.
l) Sort a result set by one or more of its columns in ascending or descending order.
m) Control or stabilize the output rate at which events are expressed in output, and suppress output events.
n) Perform relational, logical, and arithmetic operations.
o) Identify the target object of a specific sensor.

4.2. Language Design

The Domain Analysis stage is followed by Language Design stage, in which the language elements are described independently of any particular representation or encoding, i.e., its abstract syntax. The elements that comprise the BIMSL language are organized in groups that are presented in Table 1. In addition, we define the representation for language elements and production/composition rules, along with a definition of the expected behaviour of language elements.

After identifying elements of the BIMSL language, the next step is to define their particular representation. Thereafter a concrete language syntax can be built in order to fully understand the relationship between BIMSL language elements [45]. In addition, semantics may be assigned to the language elements [46].

With respect to operators, data types, and aggregation functions, the representation chosen is the most typical for query languages and programming languages in general, and is very similar to SQL. More specifically, an Identifier element starts with a capitalized or non-capitalized character from A to Z, or an underscore, followed by any number of the referenced group of characters or any digit. The Decimal element is composed of two or more digits separated with a dot or a comma. Finally, the String element comprises any quoted sequence of characters. The representation for the remaining language elements is nearly immediate. Some elements have several representations in order to achieve better usability.

Based on this representation, it was possible to build a grammar for BIMSL and observe how the language elements are related to each other. The grammar specification was built using EBNF notation, as the tool that generates the parser code takes a grammar written in this notation as input. The complete grammar specification of BIMSL is shown in Figure 2. After finishing this specification, it was possible to establish a set of rules and constraints that defines BIMSL semantics. Also, there are several restrictions that could not be expressed using the grammar, but are also related to BIMSL semantics. These rules and constraints are mostly related with the default behaviour of language elements and type of arguments.
<table>
<thead>
<tr>
<th>Group</th>
<th>BIMSL Language Element</th>
<th>Description</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clauses</td>
<td>Selection, Table Listing, Filtering, Grouping, Aggregation Filtering, Sorting, Output.</td>
<td>Starts a specific clause.</td>
<td>Enables partitioning a BIMSL query in several parts, making it easier to understand the purpose of each one.</td>
</tr>
<tr>
<td>Selectors</td>
<td>Column Naming, Distinct.</td>
<td>Selection of columns.</td>
<td>Simplifies the output presented to the user.</td>
</tr>
<tr>
<td>Data Types</td>
<td>Boolean, Decimal, Identifier, Integer, String.</td>
<td>Data type elements that can be expressed in BIMSL language.</td>
<td>Provides the most common data types.</td>
</tr>
<tr>
<td>Filters</td>
<td>Containment, Distance, Exists, In, Like, Position, Target.</td>
<td>Expresses the conditions used in order to extract only the records that fulfill a certain criterion.</td>
<td>Enables the user to compose specific conditions regarding BIM and sensors.</td>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>Addition, Subtraction, Multiplication, Division, Modulo.</td>
<td>Computes a single numerical value as result of applying the corresponding arithmetic operation.</td>
<td>Enables specifying complex conditions regarding BIM and sensor data. Supported by several query languages, like Structured Query Language (SQL).</td>
</tr>
<tr>
<td>Logical Operators</td>
<td>And, Not, Or.</td>
<td>Typical logical operations.</td>
<td>Same as above.</td>
</tr>
<tr>
<td>Relational Operators</td>
<td>Equal, Not Equal, Greater Than, Greater Equal, Less Than, Less Equal.</td>
<td>Tests the relationship between two values (operands).</td>
<td>Same as above.</td>
</tr>
<tr>
<td>Sorters</td>
<td>Ascendant, Descendant.</td>
<td>Elements related to the sorting of a result set, more specifically the elements that allow sorting the output in ascending or descending order according to a list of attributes.</td>
<td>Similarly to most query languages, BIMSL should support sorting the output presented to the user, since it deals with large sets of data.</td>
</tr>
<tr>
<td>Windows</td>
<td>Event, Second, Minute, Hour, Window, Batch.</td>
<td>Specifies windows in order to bound a data stream or to control the output rate.</td>
<td>Due to the real-time nature of sensor data, the user might specify windows and conditions using real-time units. For sake of simplicity, only seconds, minutes, and hours can be specified. This group of elements was motivated by Event Processing Language (EPL).</td>
</tr>
<tr>
<td>Aggregation Functions</td>
<td>Average, Count, Maximum, Minimum, Sum</td>
<td>Enables performing calculations on sets (groups) of data, returning the result value of executing the given function. This value can be selected as result or used as part of an aggregation filter.</td>
<td>Large sets of data must be aggregated in order to be analysed.</td>
</tr>
<tr>
<td>Output Control</td>
<td>Output All, Output First, Output Last, Output Snapshot, After Window, Every Window.</td>
<td>Elements used to control or stabilize the rate at which events are output and to suppress output events.</td>
<td>These language elements enable controlling the output of BIMSL queries. They are motivated by EPL queries and can be regarded as a nice-to-have.</td>
</tr>
</tbody>
</table>

Table 1: Summary of BIMSL language elements. These elements have been defined by means of the concepts, models, and requirements identified during Domain Analysis (see Section 4.1).
Figure 2: Representation of the complete grammar specification of BIMSL in EBNF notation.
4.3. Implementation

The language implementation stage comprises the integration of DSL artefacts, resulting from the language design stage, in a target platform [42]. Accordingly, the necessary transformations of DSL code [42] written in BIMSL concrete syntax to domain model objects that will be evaluated in the BIMSL data evaluation component are also implemented. However, in this stage the focus will be not only on BIMSL language but also on its target platform implementation, the BIMSL API, which allows installing BIMSL queries. The BIMSL API is implemented in Java and was developed using the Eclipse Integrated Development Environment. Its architecture is depicted in Figure 3 and consists of three layers: Data Acquisition layer, Data Processing layer, and Data Presentation layer.

4.3.1. The Data Acquisition Layer

The Data Acquisition layer is represented by all sources of data that may interact with the Data Processing layer. The data sources include not only streaming data produced by sensors, but also BIM models in IFC format and other static information that may be provided by the user.

In contrast with streaming data, BIM models in IFC format are persisted in a Database Management System (DBMS). This DBMS is managed on the BIMserver platform, which allows storing, maintaining, and querying building information in IFC format. In addition, a user may provide specific sensor information that BIM models do not support by editing a sensor configuration file that is interpreted by the BIMSL API. This configuration is then used in the evaluation of BIMSL queries and may include (i) the sensor type, (ii) its range, (iii) its changing frequency in milliseconds, and (iv) its target id.
4.3.2. The Data Processing Layer

The Data Processing layer provides the core functionality of the BIMSL API. It is responsible for the integration of data from several sources and the evaluation of BIMSL queries according to data acquired and BIMSL semantics.

First, the BIMSL Manager module handles all commands that come from the BIMSL Command-Line Interface (CLI). In addition, by managing the communication with the BIMserver [3, 4] and the Esper [47] real-time query engine, the BIMSL Manager is responsible for (i) installing and managing queries, (ii) handling new events and producing query output, (iii) managing a BIMserver client, and (iv) reading configuration and script files.

Since a BIMSL query has a dynamic component, related to real-time data, and a static component, related to BIM, the corresponding EPL query must be synthesized automatically and fed to the Esper engine. Therefore, the Esper engine will process only the dynamic component that is extracted by the BIMSL Parser module. Finally, once installed, the EPL query will produce events that will be handled by the BIMSL Manager.

The BIMSL Parser module is responsible for parsing a BIMSL query and defining its behaviour. In order to generate BIMSL artefacts, we used an open-source parser generator tool named JavaCC [48]. This tool takes as input the BIMSL grammar in EBNF notation and generates a Java program that recognizes matches to the grammar. Along with the grammar specification, JavaCC allows defining the semantics of BIMSL. This program is part of the BIMSL Parser module, having been successfully integrated with the BIMSL API. In addition, the parsing operation of a BIMSL query will return an object of QueryObject class. Above all, this object allows (i) building the query result based on new events processed by the BIMSL API and (ii) writing an EPL query based on its corresponding BIMSL query.

4.3.3. The Data Presentation Layer

The Data Presentation layer is the entry point of the BIMSL API. The user interface has been developed as a CLI for BIMSL API, following the example of the most typical target platforms of query languages. This allows for better user familiarity and fast learnability of BIMSL API functionality, since the commands have been designed to be clear and simple. Thus, a user can not only install BIMSL queries, but also manage complementary information, like configuration files, BIMserver projects, scripts, input, and output.

5. BIMSL Application and Evaluation

In order to validate that our solution is suitable for the integration between BIM and sensor data, several literature based queries were implemented. An example implementation query that comprised the case study is shown in Figure 4. Regarding dynamic sensor data, a simulator of data sources was used to simulate measurements from several sensors by applying the Modbus communication protocol [49]. Thus, resorting to a database of measurements, the simulator mimics the behaviour of sensors. This simulation process is consistent with real production of data streams. The data provided by the simulator are inserted into data streams and forwarded to the DSMS.

Moreover, in order to validate that our approach has managed to meet its requirements and goals, the BIMSL language was evaluated by its target users. Since the aim of our approach was to simplify the development of applications integrating BIM with sensor data, we performed evaluation sessions with software developers.

5.1. DSL Evaluation Requirements

Overall, the evaluation of DSLs is not much different from the evaluation of regular user interfaces [50, 51, 52, 38, 53, 54, 55, 56, 57]. Indeed, most requirements related to the evaluation of user interfaces are associated with a qualitative software characteristic called usability [50, 51, 52, 38, 53, 54, 55, 56, 57], which is defined by quality standards in terms of achieving the Quality in Use [58]. More specifically, the ISO 9241-11 Standard defines usability as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” [59]. Moreover, ISO/IEC 25010 expands this definition with the notion of Quality in Use, which is related to the quality perceived by the user during an actual utilization of a product in its real context of use [58].

By following an evaluation methodology for DSLs rooted on the literature, it was possible to evaluate the achieved Quality in Use of our DSL [50, 51, 52, 53, 55, 57]. More specifically, Barišić et al. find it relevant to assess the following attributes of usability and flexibility: (i) effectiveness, (ii) efficiency, (iii) satisfaction, and (iv) accessibility [50, 51, 52]. In order to evaluate BIMSL usability, we compare BIMSL with a GPL alternative that allows performing the integration of BIM and sensor data under the same conditions. Since most solutions have been developed using GPLs, and many solutions integrating BIM with real-time data are implemented in Java [14, 29, 18], we chose Java Program-
ming Language as the alternative language to be used in the usability evaluation of BIMSL. Thus, the participants should use a Java API that offers the fundamental abstractions covered by the BIMSL API.

5.2. Evaluation Process

The BIMSL language evaluation process is summarized in Figure 5. Following the literature approach, the process begins with Subject Recruitment, comprising tasks related to target user analysis, its description, and grouping into clear categories. Then, the Task Preparation activity is performed in order to determine the future tasks that need to be executed by the user and by the Language Engineer in the following activities. The next step is to perform the Pilot Session with a Domain Expert, so the training and evaluation material can be checked. This allows for identifying procedure errors and correcting them before the actual evaluation session. After everything is checked, the Evaluation Session activity is performed, which involves a subject background questionnaire, training sessions, exams, and debriefing questionnaires for each group. Thus, the subject background questionnaire is answered before the first Training Session. This Training Session introduces the language and its features to the participants, while the Exam involves writing queries based on sentences in order to collect data to be analysed in Results Analysis. A comparison of the three queries required to be implemented in the Exam can be visualized in Table 2.

The Evaluation Session ends with a Debriefing Questionnaire to be answered by the participants, whose goal is to obtain their satisfaction with the language being evaluated and assess accessibility. The questionnaire is presented in Table 3. These activities are performed for each subject group. Since we compared BIMSL language with an alternative, each group had an extra Evaluation Session, involving again Training Session, Exam, and Debriefing Questionnaire dedicated to the alternative language. The evaluation process finishes with Results Analysis.

SELECT q.bId, AVG(m.measurement)
FROM (Measurements WINDOW BATCH OF 10 Minutes) AS m,
     (SELECT t.globalId AS tId, b.globalId AS bId
      FROM IfcDistributionControlElement t, IfcBuildingStorey b
      WHERE located_at(t, b)) AS q
WHERE m.id = q.tId
GROUP BY q.bId

Figure 4: A BIMSL query for “return the average temperature measured on each floor over the last 10 minutes.”

<table>
<thead>
<tr>
<th>Query #</th>
<th>Lines of Code BIMSL</th>
<th>Lines of Code Java</th>
<th>Keywords BIMSL</th>
<th>Keywords Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query 1</td>
<td>3</td>
<td>25</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td>Query 2</td>
<td>4</td>
<td>31</td>
<td>10</td>
<td>43</td>
</tr>
<tr>
<td>Query 3</td>
<td>4</td>
<td>36</td>
<td>12</td>
<td>51</td>
</tr>
</tbody>
</table>

Table 2: Comparison of the number of lines of code and keywords used in the queries required to be implemented by the participants in the Evaluation Session, according to a standard implementation.

Following the target user profile, we obtained 18 software engineers willing to participate in the experiment (6 informed participants and 12 uninformed participants). Before starting the first training session, we asked the participants to answer a subject background questionnaire that allowed us to detail their profile. We concluded that all participants have a degree in Computer Science and basic knowledge of Java and SQL, their ages range between 23 and 36, and they have one to eleven years of experience in their field of activity. Then, the participants received some training about BIMSL and the Java API used in the experiment. In each training session, the (i) purpose, (ii) syntax, and (iii) three examples of BIMSL or the Java API have been presented.

6. BIMSL Evaluation Results

This section presents the BIMSL evaluation, following the process described in Section 5. We have analysed the following attributes:

1. Effectiveness. Determines the accuracy and completeness of the implementation of sentences [50, 51], and is assessed by analysing the results in terms of the errors produced by each participant while using each language.
2. **Efficiency.** States the effectiveness level that is achieved at the expense of resources, such as mental and physical effort [50, 51]. It is measured in terms of the time spent by the participants on each task.

3. **Satisfaction.** Expresses freedom from inconveniences and positive attitude when the language is used in a specified context of use [50]. It is assessed using a debriefing questionnaire in which the participant performs a self-assessment of his confidence level.

4. **Accessibility.** Determines learnability and memorability of language terms [50]. This attribute is also assessed through a questionnaire.

6.1. **Effectiveness of BIMSL vs. GPL.**

The effectiveness of a DSL can be assessed by analysing the error rate of the participants while using a DSL and a GPL. We start from the same exact queries for each Exam and we classify each answer of a query as essentially correct or essentially incorrect.

The error rate related to the GPL is presented in Figure 6. We conclude that informed participants had a better performance. Most of the incorrect answers are related to a large number of language and data errors. Factors like not being familiarized with the Java API and not having any sort of feedback of system execution might have been crucial to these results.

The results are more promising for BIMSL Exam tasks, as seen in Figure 6. The informed participants

<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Satisfaction</strong></td>
<td></td>
</tr>
<tr>
<td>Question 1</td>
<td>How do you classify your performance in Query 1?</td>
</tr>
<tr>
<td>Question 2</td>
<td>How do you classify your performance in Query 2?</td>
</tr>
<tr>
<td>Question 3</td>
<td>How do you classify your performance in Query 3?</td>
</tr>
<tr>
<td>Question 4</td>
<td>If you had to implement a solution requiring the integration of BIM and sensor data, would you consider using this language?</td>
</tr>
<tr>
<td><strong>Accessibility</strong></td>
<td></td>
</tr>
<tr>
<td>Question 5</td>
<td>Is it easy to specify queries using this language?</td>
</tr>
<tr>
<td>Question 6</td>
<td>Is it easy to understand the meaning of language elements?</td>
</tr>
</tbody>
</table>

Table 3: Satisfaction and Accessibility questions of the Debriefing Questionnaire. These questions are answered by the participants at the end of each evaluation session.
had almost a perfect performance, which is explained by their knowledge of BIM and IFC concepts. Furthermore, the uninformed participants still had a good performance. BIMSL queries involve writing small portions of code and when compared to Java, the participants made fewer mistakes.

Considering the total Exam score for each participant, significant differences in effectiveness can be seen in the statistical analysis for effectiveness, which is presented in Tables 4 and 5. The score is assigned to each answer according to the number of errors, and a higher score indicates a better performance. The ranks for the Wilcoxon Signed Ranks Test are in Table 4. These ranks provide the comparison between Java and BIMSL Exam scores for each participant. According to Table 4:

- There is only one case in which a participant had a better score on the Java Exam, which corresponds to the negative rank.
- For the remaining ranks, the BIMSL Exam score is always higher except for three cases in which there is a tie between the Java Exam score and BIMSL Exam score.

Finally, by considering a significance level of 5%, we computed the Z value and the asymptotic significance that corresponds to our p-value. Both values can be seen in Table 5. By considering that (i) the mathematical Z table and our Z score and (ii) the p-value are smaller than the alpha corresponding to the significance, we can conclude that:

- There is a statistically significant difference between BIMSL and Java in terms of user’s effectiveness.
- The null hypothesis H1_null is rejected.
- Since the ranks presented in Table 4 clearly show that using BIMSL is associated with a higher score (i.e., a smaller error rate), hypothesis H1_alt is accepted, meaning that a user can specify queries integrating BIM with sensor data more effectively when using BIMSL than when using GPLs.

6.2. Efficiency of BIMSL vs. GPL

The efficiency of BIMSL can be assessed by analysing the time spent by each participant in every
Exam task and comparing it with the time spent in the corresponding Java Exam task. Using the measurements of elapsed time we compute the total time spent on each Exam. The results are presented in Figure 7 for each type of participant and language. Looking at these results, it is clear that it is more efficient to use BIMSL than Java GPL. This is explained by the large portions of Java code that the users had to write in the Java Exam, in contrast with the BIMSL Exam.

The statistical analysis for efficiency is also presented in Tables 4 and 5, in which the statistically significant differences in efficiency can be seen. We compute the ranks for the Wilcoxon Signed Ranks Test, which are presented in Table 4. These ranks provide the comparison between the Java and BIMSL Exam times spent for each participant. We can see that there is not a single case in which a participant spent more time on the BIMSL Exam, i.e., positive ranks. Finally, by considering again a significance level of 5%, we compute the Z value and the p-value, which are seen in Table 5. By considering that (i) the mathematical Z table and our Z score and (ii) the p-value are much smaller than the alpha corresponding to the significance, we can conclude that:

- There is a statistically significant difference between BIMSL and Java in terms of user’s efficiency.
- The null hypothesis H2 null is rejected.
- Since the ranks presented in Table 4 clearly show that using BIMSL is associated with lower time measurements, hypothesis H2 alt is accepted, meaning that a user can specify queries integrating BIM with sensor data more efficiently when using BIMSL than when using GPLs.

### Table 4: Computed ranks for the Wilcoxon Signed Ranks Test on effectiveness, efficiency, satisfaction, and accessibility. (\(^*)\) BIMSL < Java. (†) BIMSL > Java. (‡) BIMSL = Java.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Effectiveness</th>
<th>Efficiency</th>
<th>Satisfaction</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>3.265(^*)</td>
<td>3.728(^†)</td>
<td>2.736(^*)</td>
<td>0.000(^‡)</td>
</tr>
<tr>
<td>Asymptotic Significance (2-tailed)</td>
<td>0.001</td>
<td>0.000</td>
<td>0.006</td>
<td>1.000</td>
</tr>
</tbody>
</table>

### Table 5: Wilcoxon Signed Ranks Test statistics for BIMSL effectiveness, BIMSL efficiency, participant satisfaction, and BIMSL accessibility. (\(^*)\) Based on negative ranks. (†) Based on positive ranks. (‡) The sum of negative ranks equals the sum of positive ranks.

Assessing the users’ confidence level on their performance, also allows drawing conclusions on their satis-
Table 6: Results of the questions of the Debriefing Questionnaire regarding satisfaction. These questions are specified in Table 3. Each answer follows a Likert scale.

<table>
<thead>
<tr>
<th>Question</th>
<th>Java</th>
<th>BIMSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Question 2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Question 3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Question 4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 7: Results of the questions of the Debriefing Questionnaire regarding accessibility. These questions are specified in Table 3. Each answer follows a Likert scale.

<table>
<thead>
<tr>
<th>Question</th>
<th>Java</th>
<th>BIMSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 5</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Question 6</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

The statistical significance analysis for satisfaction presented in Tables 4 and 5 shows the statistically significant differences in the confidence level of the participants and considers the sum of rates for each participant in questions regarding satisfaction. Table 4 shows that there are mostly positive ranks, i.e., cases in which the BIMSL total rate score in questions regarding satisfaction is higher than the corresponding Java total rate score, for the same participant. Finally, by considering the usual significance level of 5%, we computed the Z value and the p-value. Both can be seen in Table 5. By considering that (i) the mathematical Z table and our Z score and (ii) the p-value are much smaller than the alpha corresponding to the significance, we can conclude that:

- There is a statistically significant difference between BIMSL and Java in terms of user’s confidence.
- The null hypothesis $H_{3\text{null}}$ is rejected.
- Since the ranks presented in Table 4 clearly show that using BIMSL is associated with higher confidence, hypothesis $H_{3\text{alt}}$ is accepted, meaning that a user can specify queries integrating BIM with sensor data with more confidence when using BIMSL than when using GPLs.

6.4. Accessibility of BIMSL vs. GPL

The accessibility of BIMSL is also evaluated through the Debriefing Questionnaire in the last two questions. An analysis using simple statistical parameters for every question regarding accessibility is presented in Table 7. The statistical significance analysis for accessibility is presented in Tables 4 and 5, showing that the p-value is greater than the alpha and so there is no statistically significant difference between Java and BIMSL in either question.

Indeed, it should be noted that, despite the good average score in Question 5, there is still more room for improvement (through training). In contrast, Question 6, which is related to the learnability of language elements, had high scores for both languages, which is a success, particularly for BIMSL. Even with limited experience in the BIMSL language, the participants found it as easy to learn and to specify queries as an API implemented in a language that they were already used to working with.

7. Conclusions

The integration of BIM and sensor data is still performed in a very adhoc way using GPLs. Indeed, despite recent efforts by the scientific community, no generic approach has been proposed to overcome the engineering challenges faced by software developers when developing applications that require the integration between BIM and sensor data.

DSLs have been known for their substantial gains in expressiveness and ease of use when compared with GPLs [37]. The development of a DSL, along with its domain abstractions, enables one such generic approach, simplifying the development of applications requiring this integration. However, developing a DSL that performs an integration of BIM and sensor data is

14
a complex undertaking, requiring, besides the definition of the language, the development of a target platform, on which the queries expressed in the new language can be installed.

The DSL approach described in this article, named BIMSL, was developed using a consistent iterative methodology comprising five stages: Domain Analysis, Design, Implementation, Deployment, and Evaluation. The BIMSL evaluation stage allowed drawing several conclusions. Regarding effectiveness, it was observed that users (developers) have better performance when using BIMSL, making fewer mistakes when compared to the GPL alternative. Regarding efficiency, the gains obtained point to a much lower cognitive and manual work effort of BIMSL in comparison to the GPL alternative. Overall, results show that using BIMSL is less time consuming, even considering that BIMSL was a language that the users had just learned. Moreover, satisfaction of the users was evaluated through a standard self-assessment questionnaire, showing that the users were much more confident in their performance when using BIMSL than when using Java. Regarding accessibility, users found the specification of queries using BIMSL as easy as using a language that they are used to working with, which is yet another merit for a newly introduced language like BIMSL.

As a final remark, it should be mentioned that although the number of survey respondents is enough to ensure large effects statistical inferences with a statistical power of 0.8 at a significance level of 0.05 [61], further tests with a larger number of respondents, are recommended. Also, since tests were performed in a laboratory setting, field tests should also be conducted to quantify in objective terms the time and/or cost benefits imparted by BIMSL in industry settings.

Clearly BIMSL influences the productivity of users (typically developers) who are performing an integration of BIM and sensor data by simplifying the specification of queries that require the integration of BIM and sensor data, when compared to the alternative use of DSLs in domains such as Automotive, Aerospace, or Telecom industries translates well to the field of BIM and associated applications such as Facilities Management.

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), with reference UID/CEC/50021/2013. DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012). The authors would like to express their gratitude to anonymous reviewers for their valuable suggestions.

References
