BIMSL: A Domain Specific Language for Integrating Building Information Models with Sensor Data

Miguel Alves
Instituto Superior Técnico, Universidade de Lisboa, Portugal
miguel.bastos.alves@tecnico.ulisboa.pt

ABSTRACT
The upsurge of interest in digital models of buildings combined with an increasing sensorization of spaces is prompting for the integration between Building Information Modeling (BIM) and real-time data of sensors, which is still performed in a very ad-hoc way using General Purpose Languages (GPLs). Indeed, a standard solution to explore sensor data in the context of BIM models is missing in literature. Although several integration approaches have been proposed by the scientific community, some of them remain theoretical, while others target very specific application domains, and therefore the development of applications requiring this integration is associated to a tremendous complexity.

Herein, we overview existing approaches to the integration of BIM with sensor data, clearly identifying the gaps on research background, and aiming at the development of a Domain-Specific Query Language that, by providing substantial gains in expressiveness and ease of use, simplifies the development of applications that require processing data of sensors with complex conditions over the BIM model. Our solution is validated with experienced software developers, according to an evaluation methodology focused on usability and flexibility attributes. The results outperformed existing alternatives, indicating that our proposal is effective to integrate BIM with sensor data.

1. INTRODUCTION
The evolution of the Internet of Things is leading to an increasing sensorization of physical spaces [50]. As a result, an increasing number of applications is required to reason about their surrounding environment [59]. These applications could greatly benefit from existing information available in BIM models [50], which allows to digitally represent physical and functional characteristics of physical spaces [9, 10, 52]. Despite being highly regarded by the scientific community [57], BIM models do not provide yet capabilities to process real-time data [52], which may be provided by sensors in smart environments. As a consequence, there is a growing demand on applications capable of integrating digital building models with real-time data, which is evidenced by the increasing number of BIM applications and purposes [57]. By integrating BIM models with real-time data, these applications can reason about their surrounding environment using its digital representation.

There is, however, a plethora of challenges to overcome in order to integrate real-time data with BIM models. First, one of the main challenges is to retrieve specific information out of BIM [36, 37], particularly from Industry Found-
nism that allows to query a BIM model, and (v) an interface where the user may specify BIMSL queries. Therefore, by exploiting information provided by BIM models, BIMSL allows the specification of real-time complex queries.

Domain-Specific Languages (DSLs) have an overall favorable reputation. Indeed, DSLs have been developed over the years regarding an huge diversity of application domains to successfully bridge research gaps [38]. For example, and related with the work described herein, Mazaira and Beetz proposed a Domain-Specific Query Language named Building Information Model Query Language (BIMQL) in order to retrieve IFC-based information of BIM models [36, 37]. This language has a growing acceptance and is considered as a standard solution for retrieving static information of BIM models. However, BIMQL does not address the requirements of real-time data.

Since DSLs provide domain abstractions and gains in expressiveness and ease of use when compared with GPLs [38], software developers creating applications requiring the integration of BIM with sensor data can use BIMSL to easily overcome development challenges. However, developing this DSL is not a trivial task, since it comprises the integration of two complex and wide domains, namely BIM and real-time data.

In order to demonstrate that the proposed solution is adaptable and suitable to a diversity of application domains, a Case Study consisting of the implementation of several literature-based queries requiring the integration between BIM and sensor data was performed. In addition, BIMSL was validated according to a DSL usability evaluation recommended by the scientific community. This methodology allowed to assess the effectiveness, efficiency, and satisfaction of the user when using BIMSL against an alternative. Furthermore, it also allowed to assess the accessibility of BIMSL.

2. BACKGROUND

The background of concepts underlying this work is diverse. In this section, we start by surveying the relevant features of BIM, followed by a review of the concepts of real-time data, including the most important topics related with data streams, Data Stream Management Systems (DSMSs) and sensor networks.

2.1 Building Information Modeling

BIM has emerged as one of the key streams in construction and civil engineering within the last decade, receiving a considerable amount of attention by researchers [57]. The term has been adopted within the Architecture, Engineering, and Construction (AEC) industry by replacing Computer Aided Design, whose emphasis was focused on the representation of a building geometry using basic geometric entities [23]. Eastman et al. defines BIM as one of the most promising developments in the AEC industry, allowing the digital construction of accurate virtual building models [18]. Likewise, the National Institute of Building Sciences presents BIM as “an improved planning, design, construction, operation, and maintenance process using a standardized machine-readable information model for each facility, new or old, which contains all appropriate information created or gathered about that facility in a format usable by all throughout its lifecycle” [12].

Along with BIM also emerged the concept of BIM model, which is the digital representation of physical and functional characteristics of a facility, serving as a reliable shared knowledge resource for information about that facility [52]. While BIM models represent digitally all real-world elements and characteristics of a building, a BIM schema prescribes an argument on how each element of the building is described. Thus, a BIM schema is a standardized data structure that allows distinct tools to create and interpret a BIM model [40].

Exchange formats facilitate the flow of information between applications with different internal schemas. The most commonly recognized exchange format is IFC [13, 52]. IFC is conceived to contain physical aspects about Building Management Systems (BMSs) through the definition of IfcBuildingControlsDomain schema, which contains all the classes related to automation nodes such as sensors. However, the constructs related with this schema just allow the specification of abstract types and textual descriptions, lacking specific information about how they are programmed and, therefore, cannot cover the real-time data requirements.

2.2 Real-Time Data

Several applications take advantage from modeling data as transient streams [5] that are continuously delivered in sequence to be processed on the fly. A data stream can be defined as an ordered sequence of tuples that can only be read once or a small number of times using finite computing and storage capabilities [19]. These tuples correspond to the observation of events or variables, like network measurements, sensor readings, and so on. However, the continuous arrival of tuples is characterized as time-varying, mostly unpredictable, and unbounded [5]. Additionally, the sources of data streams are described as open-ended, flowing at high speed, and generated in dynamic environments [19]. Finally, it is well known that traditional Database Management Systems (DBMSs) are not optimized to continuously load tuples into the database, mostly because their query engine is not prepared to support the class of queries required by data stream applications [22].

To achieve an efficient processing of data streams, a new class of data processing systems known as Stream Processing Engines (SPEs) has been developed [48]. A DSMS is a type of SPE that aims at processing several input data streams from a wide diversity of sources [5] and produce a new output stream as result [5]. DSMSs can be characterized as an evolution of DBMSs in order to support the processing of data streams. As a result of the intrinsic nature of data streams, a DSMS must meet the following requirements [22]:

1. The data model and query semantics must enable operations based on the order of arrival of tuples and deflect specific sequences of tuples.
2. Due to the impossibility of storing a complete data stream in memory, the DSMS should use approximate summary structures.
3. Operators that require to process all input data streams before start producing results cannot be used. Instead, blocking operations must be bound by a window.
4. Streaming data processing should cope with intermittent data sources.
5. To ensure scalability of the system, it may be necessary the shared execution of several continuous queries,
which correspond to a class of queries evaluated continuously [5], as data streams are constantly arriving.

3. RELATED WORK

Several approaches have been proposed by the scientific community in order to combine BIM with real-time data of sensors. First, in order to understand the current state-of-art of the integration between BIM and real-time data, a comparison between the most relevant solutions is presented. Finally, a brief analysis on the most important BIM querying approaches over the last few years is performed.

3.1 Real-Time Data Applications to BIM

Given the success that BIM has experienced regarding design and construction phases, efforts to transfer information to the operation phase, i.e., real-time data, is still in its infancy [55]. The use of BIM models integrated with monitoring systems is a relatively new usage of BIM [44]. So far, several approaches regarding the integration between BIM and sensor data have been proposed [44]. However, the potential of BIM-Sensor technology has not been fully explored, particularly on application domains related with smart buildings and Augmented Reality. Most approaches regarding the integration between BIM and sensors remain just theoretical, i.e., do not have a proper implementation, and are highly dependent specifications on a specific domain, not being easily generalized. Therefore, the lack of a standard solution that simplifies the creation of applications requiring this integration is obvious, being supported by the comparison presented in Table 1, where we classified each approach analyzed according to its implementation. We can observe that most approaches have been implemented using GPLs, while the remaining solutions have not been implemented yet.

3.2 BIM Querying Approaches

Over the last years, the complexity of engineering models has been increasing considerably, as well as the amount of data attached to these models. In order to deal with this large amount of data, several approaches on BIM querying have been proposed by the scientific community. The most relevant approaches include the Express Query Language [30], the Partial Model Query Language [1], the Generalized Model Subset Definition schema [54], the Georgia Tech Process to Product Modeling [32], the DSL Building Environment Rules and Analysis [34], the mvdXML format [15], the Solibri Model Checker application, and the DSL BMQL [37]. However, none of the approaches considers real-time data, which may be related with highly dependent specifications associated to a specific application domain.

Mazairac and Beetz proposed a DSL named BMQL, which is the most relevant approach regarding BIM querying, as it is aimed as 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, standing out the lack of a DSMS in the platform where BMQL is implemented and the lack of spatial query operators.

4. METHODOLOGY

The analysis performed in previous sections exposed the tremendous complexity of creating applications that require the integration between BIM and sensor data. In fact, retrieving specific information from BIM models and combine that information with the one provided by DSMSs constitute an imposing challenge. As demonstrated in Section 3, most approaches regarding this problem target a very specific domain, they are associated with the use of GPLs, or they are not implemented yet. Therefore, the development of applications requiring the integration between BIM and sensor data lacks a standard solution providing domain abstractions that alleviates the cognitive overload of these complex domains.

Since DSLs are known for their substantial gains in expressiveness and ease of use when compared with GPLs [38], for a limited domain, the proposed solution was developed as a Domain-Specific Query Language named BMISL. Its development followed a DSL development methodology supported by literature. According to Barišić et al., the development of a DSL is an iterative process composed of five stages, namely (i) Domain Analysis [8], (ii) Language Design [8], (iii) Language Implementation [8], (iv) Language Deployment [51], and (v) Language Evaluation [8]. These stages correspond to the formal definition of the DSL life cycle [51], which is summarized in Figure 1.
However, the development of the DSL was not enough, since we needed to develop the target platform where we could install queries in BIMSL language. We called this platform as BIMSL Application Programming Interface (API). The BIMSL API has the purpose of (i) taking into account an updated building model, (ii) integrating data of 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, a Case Study consisting of the implementation of several BIMSL queries was performed.

5. DOMAIN ANALYSIS

In the first stage of DSL development is defined the core language model of the DSL [49]. This stage provides the essential domain concepts and knowledge that will support the language design [45] by taking into account certain domain particularities like terms and expressions intrinsic to the problem solution [45, 51]. Therefore, the relevance of developing a vocabulary that is easy for domain experts to use is significant [35]. The Domain Analysis was conducted with a domain expert collaboration in order to identify concepts, terms, and expressions relevant to the problem solution. In addition, by means of a literature-based survey of queries combining BIM with sensor data, it was possible to express required characteristics that the DSL must provide.

The integration between BIM and sensor data is the union of two complex and wide domains. In fact, there is a large set of concepts and terms used within both areas. The following set represents the relevant concepts to the problem solution:

- **BIM Model.** The information model that represents characteristics of a facility using IFC format. The BIM model defines IFC objects, relationships among the objects, and object properties that will be processed in order to get a query result.

- **Sensor.** Detects events or changes in its environment and provides real-time data to the system. It has a representation in the BIM model and it is unequivocally identified by its id.

**Stream.** An ordered sequence of tuples of data. It is managed by a DSMS and, in our case, must be windowed.

**Measurements.** A set of tuples that establish a window, each one represented by the sensor id and its respective measurement.

**Window.** Allows to bound a data stream using a number of events or units of time as a limit, since it is unfeasible to store entire data streams.

**Containment.** The containment relationship of an element within a spatial structure. This relationship allows to establish the relative location of a specific object.

**Distance.** The distance between two sensors, according to their respective positions.

**Position.** The object being sensed.

In the end of this stage, it was possible to conclude that the BIMSL queries may:

1. Process a BIM model and its objects.
2. Process real-time data that is provided by sensors, by using a DSMS. The query received by the system must be transformed into an Event Processing Language (EPL) query that will deal with real-time data processing.
3. Select attributes or properties from IFC objects.
4. Select measurements, which are bounded by a window.
5. Apply aggregation functions on a result-set.
6. Build a table with attributes, properties, and measurements bounded by a specified window as columns. Cartesian Product may be applied.
7. Apply filters to records and extract only the records that fulfill a specified criterion.
8. Group a result-set by one or more of its columns.
9. Apply filters using aggregation functions.
10. Sort a result-set by one or more of its columns, in ascending or descending order.
11. Control or stabilize the rate at which events are output, and suppress output events.
12. Perform relational, logical, and arithmetic operations.
13. Establish containment relationships between IFC objects.
14. Calculate the distance between two IFC objects.
15. Calculate the position of an IFC object.
16. Identify the target object of a specific sensor.

6. DESIGN

The Domain Analysis stage is followed by the Language Design. In this stage, we start by describing the language structure, including its properties, rules, and relations, i.e., its abstract syntax [45]. Then, we define the language notation, i.e., its concrete syntax [45]. The stage finishes with the definition of the expected behavior of language elements [45].
### Group BIMSL Language Elements Function

<table>
<thead>
<tr>
<th>Group</th>
<th>BIMSL Language Elements</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation Functions</td>
<td>Average, Count, Maximum, Minimum, Sum</td>
<td>Allow to perform calculations on sets 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>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>Addition, Subtraction, Multiplication, Division, Modulo</td>
<td>Compute a single numerical value as result of applying the corresponding arithmetic operation.</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>
</tr>
<tr>
<td>Filters</td>
<td>Containment, Distance, Exists, In, Like, Position, Target</td>
<td>Include any condition used in order to extract only the records that fulfill a certain criterion.</td>
</tr>
<tr>
<td>Logical Operators</td>
<td>And, Not, Or</td>
<td>Common logical operations.</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>
</tr>
<tr>
<td>Relational Operators</td>
<td>Equal, Not Equal, Greater Than, Greater Equal, Less Than, Less Equal</td>
<td>Allow to test the relation between two operands, including equalities and inequalities.</td>
</tr>
<tr>
<td>Selectors</td>
<td>Column Naming, Distinct</td>
<td>Elements related with the selection of columns.</td>
</tr>
<tr>
<td>Sorters</td>
<td>Ascendant, Descendant</td>
<td>Elements related with the sorting of a result-set, more specifically the elements that allow to sort a column in ascending or descending order.</td>
</tr>
<tr>
<td>Statements</td>
<td>Selection, Table Listing, Filtering, Grouping, Aggregation Filtering, Sorting, Output</td>
<td>Allow to partition the query in parts in a way that each one is associated with a specific purpose.</td>
</tr>
<tr>
<td>Windows</td>
<td>Event, Second, Minute, Hour, Window, Batch</td>
<td>Specification of BIMSL windows in order to bound a data stream or to control the output rate.</td>
</tr>
</tbody>
</table>

Table 2: The groups of BIMSL language elements.

6.1 Abstract Syntax

The Design stage started by applying the results of Domain Analysis in order to describe the language elements that compose the BIMSL language, independently of their particular representation. These elements are organized in several groups, presented in Table 2. Starting from this description, it was possible to build a grammar for BIMSL and observe how the languages elements are related to each other. The grammar specification was built using Extended Backus-Naur Form (EBNF) notation, since the tool that generates the parser code takes a grammar written in this notation as input. After finishing BIMSL grammar specification, we established a set of rules and constraints that defines BIMSL structural semantics. These rules and constraints are mostly related with default behavior of language elements and type of arguments.

6.2 Concrete Syntax

After describing BIMSL abstract syntax, the next step is to define the particular representation of BIMSL language elements [45], i.e., its concrete syntax. The concrete syntax of a language refers to its notation and the way users will learn and will use it, either by reading or by writing [45]. In the case of BIMSL, the notation used is only textual.

Regarding operators, data types, and aggregation functions, the chosen representation is the most typical for query languages and programming languages in general, being very similar to Structured Query Language (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 textual representations in order to achieve better usability.

6.3 Semantics

As the particular representation of BIMSL language elements is defined, semantics may be assigned to these elements [11], using code generation mechanisms. In fact, code generation can be regarded as a form of semantics [29]. By considering the meaning of BIMSL language elements and structural semantics of BIMSL, it was possible to define the semantics of our language. The semantics has been defined using JavaCC tool, which takes a grammar in EBNF notation as input and allows to put portions of Java code next to rules and language elements.

7. IMPLEMENTATION

This stage is the actual language implementation, comprising the integration of DSL artifacts resulting from the language design stage in a target platform [49]. Accordingly, the necessary transformations of DSL code written in a concrete syntax to programming language code are implemented [49]. However, in this stage the focus will not only be on BIMSL language but also in its target platform implementation, the BIMSL API, which allows to install BIMSL queries. The BIMSL API is implemented in Java.
and it was developed using the Eclipse Integrated Development Environment. Its architecture is depicted in Figure 2 and is composed of three layers.

### 7.1 Data Acquisition Layer

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

Regarding dynamic sensor data, a simulator of data sources was used to simulate measurements from several sensors by applying the Modbus communication protocol. Thus, resorting to a database of measurements, the simulator mimics the behavior of sensors. This simulation process is consistent with real production of data streams. The data provided by the simulator is inserted into data streams and forwarded to the DSMS.

In contrast with streaming data, BIM models in IFC format are persisted in a DBMS. This DBMS is managed by BIMserver platform, which allows to store, maintain and query 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.

### 7.2 Data Processing Layer

The Data Processing layer provides the most important functionality of BIMSL API, being 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 allows to handle all commands that come from BIMSL Command-Line Interface (CLI). In addition, by managing the communication with BIMserver and Esper Engine, 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 with real-time data, and a static component, related with BIM, the corresponding EPL query must be created only regarding this dynamic component. Therefore, the Esper Engine will only process the dynamic component that is extracted by BIMSL Parser module. Finally, the installed EPL query will produce events that will be handled by BIMSL Manager.

Second, besides operating a DSMS, the Esper Engine module also manages the communication with the input terminal, the output terminal, and output files. The EPL queries are installed in the DSMS, which is based on EsperTech technology.

Finally, the BIMSL Parser module is responsible for parsing a BIMSL query and defining its behavior. In order to generate BIMSL artifacts, we used an open-source parser generator tool named JavaCC [16]. 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 allowed to define the semantics of BIMSL. This program is part of BIMSL Parser module, having been successfully integrated in the
SELECT m.id, m.measurement
FROM IfcBuildingStorey b,
     IfcDistributionControlElement t,
     (Measurements WINDOW OF 1 Event) AS m
WHERE m.id = t.globalId
AND located_at(t, b)
AND b.name = "3rd Floor"
OUTPUT ALL EVERY 1 Minute

Figure 3: A possible implementation in BIMSL for the query “every minute, return temperature values measured by all sensors on the third floor”.

BIMSL API. In addition, the parsing operation of a BIMSL query will return an object of QueryObject class. Above all, this object allows to (i) build the query result based on new events processed by the BIMSL API and (ii) write an EPL query based on its corresponding BIMSL query.

7.3 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 a better user familiarity and fast learnability of BIMSL API functionality, since the commands have been designed to be clear and simple. Thus, a user cannot only install BIMSL queries but also manage complementary information, like configuration files, BIMserver projects, scripts, input and output.

8. CASE STUDY
In order to demonstrate that our solution is suitable for the integration between BIM and sensor data, a Case Study consisting of the implementation of literature-based queries was performed. This Case Study surveys the features that result from the composition of BIM and real-time monitoring of sensor networks, and it is performed in the context of Language Deployment stage, where language engineers use the DSL to specify models, resulting in working software [8]. An example of the implementation of a query that compose the Case Study is presented in Figure 3.

9. EVALUATION
In order to validate that our approach has managed to overcome its requirements and goals, the BIMSL language must be evaluated. In fact, without a proper evaluation in overall DSL development, the process is still incomplete. Therefore, BIMSL has to be validated with its target users. Since the aim of our approach is to simplify the development of applications integrating BIM with sensor data, we performed evaluation sessions with software developers. We start with a description of the evaluation methodology and its participants, followed by an analysis of the results obtained.

9.1 Methodology and Participants
By following an usability evaluation methodology based on literature, it is possible to evaluate the achieved Quality in Use of our DSL [6]. More specifically, Barisić et al. finds relevant to assess the following attributes:

- Effectiveness. Determines the accuracy and completeness of the implementation of sentences [6, 7].
- Efficiency. States the effectiveness level that is achieved at the expense of resources, such as mental and physical effort. It is commonly measured in terms of the time spent to complete a sentence [6, 7].
- Satisfaction. Expresses freedom from inconveniences and positive attitude when the language is used in a specified context of use [6].
- Accessibility. Determines learnability and memorability of language terms [6].

In order to evaluate BIMSL usability, we need to compare BIMSL with another language that allows to perform the integration between BIM and sensor data in the same conditions. Since most solutions have been developed using GPLs, and a large group of solutions integrating BIM with real-time data is implemented in Java [14, 20, 21], we chose Java Programming Language as the alternative language to be used in the usability evaluation of BIMSL.

The BIMSL language evaluation process is summarized in Figure 4, following the literature approach. The process begins with (i) Subject Recruitment, followed by (ii) Task Preparation, in order to prepare future tasks. The next step is to perform the (iii) Pilot Session with a Domain Expert, so the training and evaluation material can be checked. Then, the Evaluation Session is performed, which involves a subject background questionnaire, training sessions, exams and final questionnaires for each group of participants. A (iv) Training Session allows to introduce the language being evaluated to the participants, while (v) the Exam involves writing queries based on sentences. The Evaluation Session ends with a (vi) Final Questionnaire, whose goal is obtaining the satisfaction of the participants and their assessment on accessibility. The evaluation process finishes with (vii) an analysis of results obtained.

Finally, the objective of our evaluation experiment is to answer the following research questions:
- Is specifying queries integrating BIM with sensor data more effective when using BIMSL than when using Java?
- Is specifying queries integrating BIM with sensor data more efficient when using BIMSL than when using Java?
- Are the participants using BIMSL more confident on their performance than when using Java?
After the methodology and the goal of the usability evaluation had been defined, we needed to select the subjects to be part of this process, following the target user profile. Therefore, we identified two types of participants, according to our solution context:

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

**Uninformed participant.** They are software developers who do not have previous experience on BIM and real-time data processing, being also regular users of GPLs.

### 9.2 Results

The obtained results of our usability evaluation experiment allowed to draw several conclusions concerning each attribute:

- **Regarding effectiveness,** we concluded that the users have better performance when using BIMSL, by making less mistakes when compared to the alternative.

- **Regarding efficiency,** we confirmed that BIMSL allows to specify queries with much lower mental and physical efforts than alternative. This is demonstrated by our evaluation experiment, where we observed that using BIMSL is less time consuming, even considering that BIMSL was a language that the users had just learned.

- **Regarding satisfaction,** the self-assessment performed by the users allowed to confirm their positive attitude towards BIMSL. Indeed, the results showed that the users were much more confident on their performance when using BIMSL than when using Java.

- **Regarding accessibility,** the results showed several similarities with Java. Although neither language was significantly better than the other, these similarities indicate that the users found the specification of queries using BIMSL as easy as using a language that they are used to work with, which is a positive aspect for a recent language like BIMSL.

Our usability evaluation experiment with target users of BIMSL allowed to confirm that the language really meets its requirements. Since the analysis of results revealed successful outcomes, we have determined that, by using BIMSL, the users increase their effectiveness, efficiency, and satisfaction during query specification.

Since DSLS can be considered as user interfaces [7], our usability evaluation experiment allowed to observe how BIMSL positively influences the productivity of users that are performing an integration between BIM and sensor data. Indeed, the results determined that this solution really simplifies the specification of queries that require the integration between BIM and sensor data when compared to the alternative, i.e., the use of GPLs, and therefore we can say that our solution simplifies the development of applications requiring this integration.

### 10. CONCLUSIONS

Despite several approaches proposed by scientific community, the integration between BIM and sensor data is still performed in a very ad-hoc way using GPLs. In fact, most solutions regarding this integration target very specific application domains, like BA, EM, and Augmented Reality, or remain just theoretical, not having been implemented yet. Therefore, the development of these solutions is associated to a high level of complexity, lacking a standard solution adaptable to several application domains in order to retrieve real-time information associated with BIM models. Software developers creating this type of applications could use this standard and adaptable solution that would alleviate the cognitive overload regarding these complex domains.

In contrast with other existing approaches, our solution was designed as a standard approach to be used by software developers in the development of applications requiring the integration between BIM and sensor data. In fact, through the development of a DSL, along with its domain abstractions, it is possible to simplify the development of applications requiring this integration, since DSLs have been known for their substantial gains in expressiveness and ease of use when compared with GPLs [38]. Thus, we named our DSL as BIMSL. However, developing a DSL that performs an integration between BIM and sensor data was a difficult task, since retrieving specific information from BIM models and combining that information with the data processed by DSMSs constituted a tremendous challenge. Furthermore, both BIM and real-time data processing are wide and complex domains, and the development of BIMSL was not enough, since a target platform where we could install queries using this language was required. We named this target platform as BIMSL API.

Another merit of the DSL developed under the context of this work lies in the development methodology adopted. In fact, we followed an iterative DSL development process composed of five stages. First, in Domain Analysis stage was defined the core language model of BIMSL by identifying domain concepts and knowledge. Second, in Design stage was described the abstract syntax, concrete syntax, and semantics of BIMSL. Third, in Implementation stage, BIMSL artifacts resulting from the language design stage have been integrated in the target platform, i.e., the BIMSL API. Indeed, the implementation of BIMSL API was also conducted in this stage. Fourth, in Deployment stage, it was developed a Case Study consisting of the implementation of several literature-based queries. This Case Study allowed to demonstrate the flexibility of BIMSL and its easy adaptation to several application domains. Finally, in Evaluation Stage, the DSL was validated with its target users.

In order to perform the Evaluation stage of BIMSL development, we followed a rigorous usability evaluation experiment based on literature to assess (i) the effectiveness, (ii) efficiency, and (iii) satisfaction of the user when using BIMSL, and the (iv) accessibility of BIMSL. Therefore, we needed to compare BIMSL with another language that allows to perform the integration between BIM and sensor data in the same conditions. Since most solutions have been developed using GPLs, we chose Java as the alternative language to be used in the usability evaluation. This evaluation experiment with target users allowed to prove that, by using BIMSL, the users increase their effectiveness, efficiency, and satisfaction during query specification. The accessibility is similar with both languages. In fact, specifying a query that requires the integration of BIM with sensor data is less error-prone and time consuming when using BIMSL than when using GPLs, and therefore BIMSL undoubtedly simplifies the development of applications requiring the integration between BIM and sensor data.
REFERENCES


[31] K. P. Lam and V. Srivastava. Living in the Intelli-