C. M. Herr, N. Gu, S. Roudavski, M. A. Schnabel, *Circuit Bending, Breaking and Mending: Proceedings of the 16th International Conference on Computer-Aided Architectural Design Research in Asia,* 731–740.

USING DOMAIN SPECIFIC LANGUAGES IN THE BUILDING INFORMATION MODELLING WORKFLOW

RUWAN FERNANDO,[1] JAMES STEEL[2] AND ROBIN DROGEMULLER[3]
Queensland University of Technology, Brisbane, Australia,
1. r1.fernando@qut.edu, 2. james.steel@qut.edu.au, 3. robin.drogemuller@qut.edu.au

Abstract. The design of architecture, in practice, entails the collaboration of many disciplines each with their own set of tools and representations. Building Information Models aim to support interoperability between these disciplines. However current implementations require a lot of manual work involving translating parts from the various specialised descriptions to the common model format. Domain Specific Languages are a development from Information Technology that defines a mapping from the concepts used in one discipline to those used in another. In this paper, a workflow incorporating the movement between specialised languages and a central model is described. The central model is structured using the Industrial Foundation Classes (IFC). The motivation for elaborating on the interdisciplinary workflow is the desire to create a more iterative process without the need for the manual recreation of models. While it is difficult to have a description or language that contains all the information of all the disciplines, this research demonstrates how the IFC schema acts as a pivot not just between data sets, but also between concepts expressed in different representations thus giving from analysis to design.

Keywords. Building Information Model; BIM; domain-specific languages; lighting; spatial planning; parametric modelling.

1. Introduction

In software development, a domain-specific language (DSL) is a smaller language dedicated to a particular problem domain. Deursen et al offer the following definition:

> A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. Deursen et al (2000.)

In this regard domain specific languages are not at all new – architects use a variety of media to explore different concepts from orthographic drawing to physical models, engineers use mathematical abstractions and simulation models. Many computer-aided design tools allow for scripting or *macros*, to enhance workflow by grouping together repetitive sets of commands – these in fact form a small 'private' DSL used by the modeller.

An important distinction should be made between 'domain' and 'discipline.' The Architecture, Engineering and Construction (AEC) industry has evolved to differentiate between disciplines such as landscape design or electrical and mechanical design. A domain can be shared between several disciplines such as the domain of 'spatial reasoning' or 'geometry' or several domains can be used within a single discipline e.g. in HVAC engineering, Computational Fluid Dynamics and Duct Layout modelling are two distinct domains. Domain specific languages and modelling tend to use higher-level abstractions than a general purpose modelling language. For example consider the scripting language in Computer-Aided Design (CAD) software such as '*Rhinoscript'* in comparison to a general programming language such as *'Java.'* Thus a DSL requires less effort (for the human modeller) and fewer articulations in at the low-level aspect of computation.

Building Information Modelling (BIM) is fast gaining popularity in the AEC industry. For an explanation of BIM or its use in the industry, several authorities are available e.g. Eastman (1995). Because BIM uses three-dimensional geometric modelling as well as the representation of meta-data, it is an ideal way to move information between groups. Older forms of communication such as 2D drawings are prone to misinterpretation – and require remodelling of geometry when tasks such as simulation or visualization are needed. The largest attraction of BIM is therefore, that it makes possible an integrated approach to design – different disciplines can modify or analyse a model during its inception in opposition to the serialised approach of moving drawings from one group the next only after completion.

The formalisation of BIM that this paper elaborates on is the Industry Foundation Classes (IFC). These represent an accepted open standard that is supported by most of the major CAD vendors. Most other BIM descriptions are proprietary or limited to a single vendor. While successes have been reported in many cases with the use of IFC, there is lot of work that needs to be done with regard to connecting models to analysis and domain specific software.

For a description of how the IFC schema operates see Steel et al (2010). The main idea that this research proposes is that the building information model act as a pivot, being mapped to by different domain specific languages. The complexities and issues of this are discussed.

2. Domain-specific languages: Theory and use in the AEC industry.

At a very simple level, a DSL can be implemented through inputting text instructions in a manner similar to using *macros* available in many software systems. In this context however, it is more likely that the instructions that are sent to the computer will be in the form of interactions with a GUI (Graphic User Interface.) Almost all of the most popular CAD and BIM software available allow for extension using powerful general purpose languages.

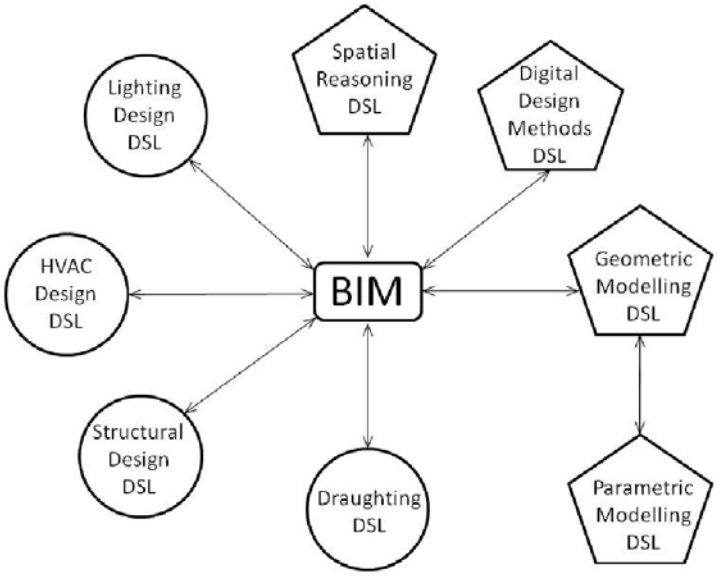

Figure 1. Different domains with a central BIM

Figure 1 isolates several common domains that are present within the architecture and engineering disciplines. The domains that are enclosed in a circle are particular to a discipline while the domains within pentagons are shared

amongst several disciplines. In the field of software engineering, using domain-specific languages in order to describe software systems, stretches back to the early 80's and the development of so-called fourth-generation languages (4GLs), but has garnered more attention recently – Fowler (2010). The essential idea is that, rather than requiring the designer to express their design model in a general-purpose language with concepts at a lower level abstraction, they be given a language whose concepts are a closer to those in which they think about the problem and solution space. Typically, these domain-specific languages can then be mapped to lower-level languages, in order that they are subject to more general-purpose tools, either to analyse the design, or in the common case for software engineering, to actualise it into a running system. The DSL approach is particularly useful when dealing with diverse domains, where the lexicon of the designer is removed from that of the lower-level languages they might otherwise be forced to employ, as well as in situations where different domain experts need to work on a unified system (Hessellund et al, 2007).There are a number of possible approaches taken to the development of a domain-specific language. The simplest is through the use of *macros*, whereby a domain-specific concept can be encoded as an in-place expansion to a number of concepts in the lower-level language. This approach, common in languages such as C++, has the advantage of being easy to implement and evolve, and of allowing the user to mix and match concepts from the DSL and the lower-level language, but requires the DSL to have a strong structural similarity to the lower-level language. Another approach is by embedding the domain-specific concepts and syntax within a general purpose language. This is common in languages such as Ruby or Lisp, and also has the advantage of giving the designer the ability to use concepts, but provides more support for domain-specific syntax than the macro approach. A third approach is to use more heavyweight tools in order to design the DSL and its supporting tools from the ground up. This allows the DSL to be structurally independent of any lower-level language, and thus greater flexibility of form and of syntax, but requires more effort to provide tool support and a mapping back to a lower-level language, although this has become dramatically easier with recent advances in language- and model-driven engineering. Given the diversity of disciplines used in AEC, and for reasons stated earlier it is this third approach that offers the most promise when looking at DSLs for the Architecture, Engineering and Construction industry.

The lower-level language serves an important role in the use of domain-specific languages. It serves as the pivot point where domain-specific designs can be integrated, and exposed to tools for simulation or analysis, thus obviating the need for the DSL developer to develop separate tools for the design model

in its domain-specific form. In this paper, we propose that the most appropriate language to serve as a pivot in the application of the DSL approach to the AEC space is that of the Industrial Foundation Classes. One of the common criticisms of IFC is that it offers too wide a variety of modelling constructs, but this can serve as a strength when it is employed as a pivot language for the integration of heterogeneous domain-specific design models. Also, as the significant open standard in the digital design space, IFC remains one of the best conduits for connecting domain-specific models to analysis tools.

The benefits of a DSL approach to design in the AEC space require that there be a sound link between the domain-specific language and the pivot language. To this end, we are exploring the use of model transformation languages, which are becoming a popular tool for mapping between different modelling languages within the model-driven engineering community. There exists a variety of open-source tools [Jouault, 2006; Steel 2004] for describing the concept-to-concept mappings between modelling languages, and for automating the conversion of their models. Since the mappings between languages can vary greatly, we anticipate using different model transformation languages for the different domain-specific languages.

3. Parametric modelling: Lessons for object-oriented programming

Flexible modelling is at the foundation of the workflow being explored. Without the ability to make changes rapidly, an integrated approach becomes unfeasible with the time constraints set in normal competitive practice. While many different opinions and definitions exist with regard to what constitutes parametric modelling, the definition used here is that in a parametric model, the geometric properties of any CAD object can be expressed as a function of other objects or values. For example if a beam were to span between two walls, bringing the walls them closer or further apart would change the length of the beam to follow this constraint. A more advanced way in which parametric modelling can be articulated is by taking an object-oriented approach. Many lessons can be learnt from the evolution of object-oriented programming concepts and methods. Some fundamental concepts used within this field are inheritance, polymorphism and encapsulation.

3.1. INHERITANCE

Inheritance is the act of one object inheriting properties from another object. An object that cannot be instantiated but only inherited from is described as '*abstract.'* This principle is illustrated in Figure 2. The top-left image shows an abstract 'jig' for a floor slab. This object defines a planar polygon and a one

dimensional grid. This abstract class can be inherited from and instantiated in several ways – the first one given as a simple slab (the IFC scheme carries the object 'IFCSLAB' within its object set) the next two are assemblies of beams and the slab. The beam definition is replaceable as long as certain criteria are met. In this case, that it be defined by a planar polygon and extruded along a straight line.

3.2. ENCAPSULATION

Along with inheritance, an important concept that follows is one of encapsulation. This aspect of object-oriented programming controls the level of access of different properties of the objects. In the floor slab example, the polygon that defines the shape of the floor might be constrained to being planar by 'hiding' the Z-coordinate of the points or lines that define the shape. Thus avoiding any scenario which a non-planar polygon is given. As the floors become more complex, more properties are made variable – with the fourth floor example, the properties of the I-beam that supports the floor become part of the assembly.

In systems involving the manipulation of complex objects, it is imperative that the methods and functions that operate on these objects are easily accessible when dealing with them. One popular method is to release these objects with their functions within the same definition. For example if the area or longest dimension of the polygon that defines the floor was needed by another operation or object, the tasks of getting this data is simplified when it is held by the object itself.

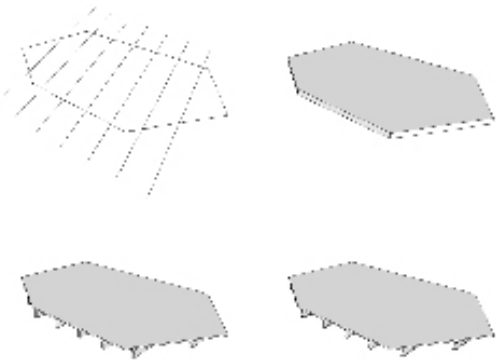

Figure 2.Floor system, inheritance from an abstract class

4. Case studies

4.1. LIGHTING DESIGN

The field of lighting makes a good introductory study in that the simulations performed by lighting analysis software such as Radiance is largely made up of only three parts. The first is the geometry of the objects in the scene, the next being the material properties of these objects and the last being luminaire definitions. The IFC scheme itself holds several lamp and luminaire definitions, so the mapping to a lighting analysis package is a straight-forward task as illustrated in Figure 3. The language used by lighting designers include words such as *'illuminance', 'lux', 'glare', 'daylight factor'* and *'emissivity.'* A unit of measure common to this field as well as of interest to energy efficient design is that of the Watt. Therefore this becomes a unit that can be used as a baseline to discuss the benefits of different variations.

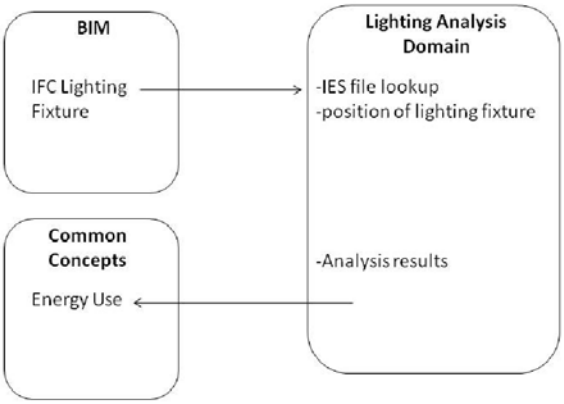

Figure 3.The lighting design domain

4.2. SPATIAL REASONING

A serious issue with a lot of BIM-led projects is the amount of data that is present in the files. This can cause long waiting periods for the model to be visible in 3D. The simple act of being able to select what should be sent to the graphics pipeline has enormous benefits in reducing the waiting time for the model to be made visible. Many CAD and BIM packages allow for the

viewing of layers – differentiated by pre-set or customized object categories. An example of a fragment of a DSL used for spatial reasoning could appear as: *'show levels 2-3, structural, and lighting'*.This would only send the selected levels to the pipeline in that view without having to manually switch on or off layers.

4.3. HVAC AND STRUCTURAL DESIGN

The Heating, Ventilation and Air-Conditioning discipline is interesting in that the common definitions used are not just geometry but also topology. It is common to describe duct layout systems using graphs – nodes, edges and weights. Ductwork layouts can be complex entities to describe procedurally in this regards as the room shape can shape the topology of connections – a weakness of parametric modelling software. In this case iterative solutions are sometimes better. Brahme and Mahdavi (2001) give a solution to the problem of formalising duct work dynamically with regard to topological changes. As environmental services represent a large amount of the buildings total energy use, it should be noted that interaction between this domain and energy analysis is a critical one.

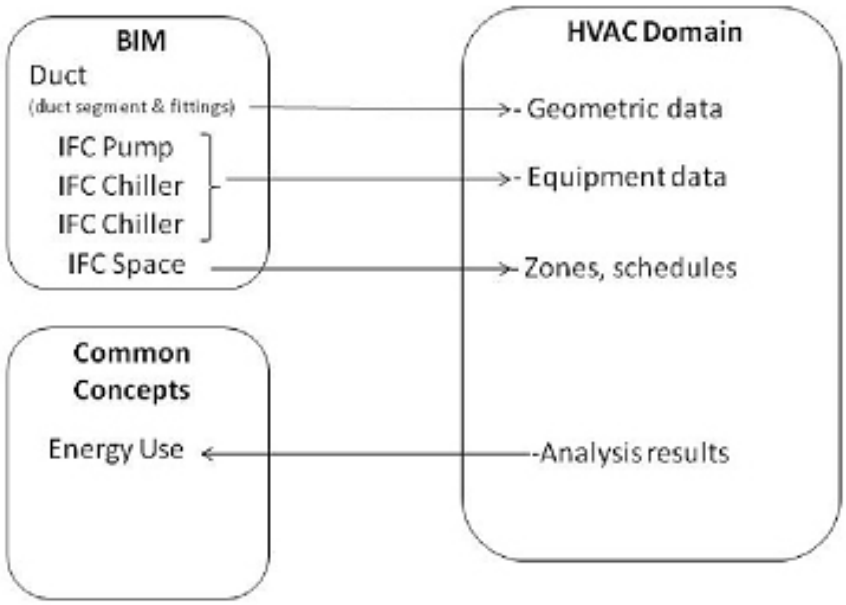

Figure 4.The HVAC domain

4.4. GEOMETRY

Geometry is a field common to all disciplines and is an important domain holistically. A more specialised DSL within geometry is that of parametric modelling, the importance of which was discussed in the previous section. The DSL for this domain, capture important relationships in having the objects adjust to different scenarios. While many texts exist entirely on this topic (i.e. Lee, 2006) for the purposes of the main ideas presented in this research, the following points are made:

- It should be possible for a BIM capable system to describe an object procedurally so that it can be rebuilt according to difference parameters.
- Objects should have multiple levels of representation appropriate to levels of detail visible.
- It should be possible to link the parameters of an object to that of another to associate the properties of objects.

Flexible geometry manipulation lies at the very foundation of creating an iterative process.

4.4. GENERATIVE DESIGN

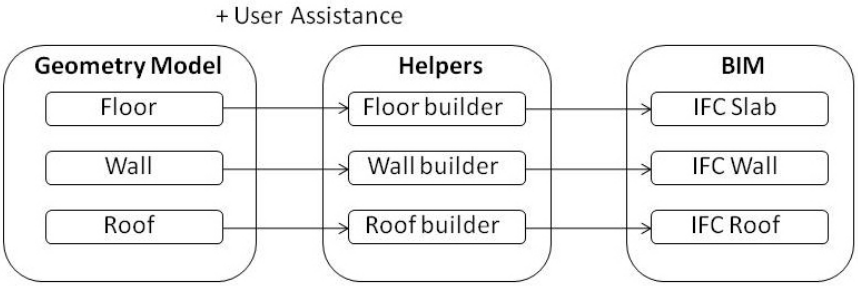

Figure 5. Creating a BIM from a geometric model

This topic was chosen for variety as it does not fit into practice, but rather, academia. The use of BIM at the level of study is less popular as most BIM software packages do not give the freedom of free-form modelling packages. There is however benefits to using BIM even at the study level – the most pertinent one being the use of analysis tools. The most important DSL linking these two fields is a language allowing for objects to be described by geometry alone – and then having their BIM definition written out. A set of 'helpers'

that formalise 'shapes' into BIM objects - floors and walls – can help move between such tools and building information models. More sophisticated design techniques are being developed in many diverse AEC related fields – for example in HVAC, Beghi et al (2009), show the use of genetic algorithms to manage the designing HVAC systems. If such systems make use of BIM as a pivot, researches would have easy access to well-developed modelling and visualisation systems.

5. Conclusion

While BIM has great potential and is quickly becoming the norm in the AEC industry, there are still many barriers to overcome. The most important of these, being able to create a feedback loop between analysis and model adjustment. As an activity, design relies on being able to work with an intuitive understanding of problems and their possible solutions. It is important to be able to break a problem into parts, but also to be able to work both holistically and in a specialised manner. The closer representations come to the conceptualization we use to solve problems, the easier it is to articulate them. In this regard, domain-specific languages are a powerful way to build on knowledge and to build systems on top of other systems.

Acknowledgements

This paper was developed from research funded in part by the Australian Research Council.

References

Beghi, A., Cecchinato, L., and Rampazzo, M.: 2011, A Multi-phase genetic algorithm for the efficient management of multi-chiller systems. *Energy Conversion and Management*, **52**(3), 1650-1661.

Brahme, R., Mahdavi A., Lam, K. and Gupta, S.: 2001, Complex Building Performance Analysis in Early Stages of Design, *IBPSA 2011*, Rio de Janeiro, Brazil, 661-668.

van Deursen, A., Klint, P. and Visser, J.: 2002, Domain-Specific Languages. In *The Encyclopedia of Microcomputers*. Marcel Dekker Inc., 53-68.

Eastman, C. M.: 2008, *BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractor*, Wiley, Hoboken, NJ.

Fowler, M.: 2010*, Domain specific languages*, Boston, MA, Addison-Wesley.

Hessellund, A., Czarnecki, K. and Wasowski, A.: 2007, Guided Development with Multiple Domain-Specific Languages. *ACM/IEEE 10th International Conference On Model Driven Engineering Languages and Systems (MODELS 2007),* Nashville, TN, USA,1-15.

Lee, G., Sacks, R., and Eastman, C. M.: 2006, Specifying parametric building object behavior (BOB) for a building information modeling system, *Automation in Construction,* **15**(6), 758-776.

Steel J. and Lawley M.J.: 2004, Model-Based Test Driven Development of the Tefkat Model-Transformation Engine, *15th Intl Symposium on Software Reliability Engineering (ISSRE 2004)*, St Malo, France, 151-160.