S. Chien, S. Choo, M. A. Schnabel, W. Nakapan, M. J. Kim, S. Roudavski (eds.), *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA 2016,* 147–156. © 2016, The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), Hong Kong.

PORTABLE GENERATIVE DESIGN FOR BUILDING INFORMATION MODELLING

S. FEIST, G. BARRETO, B. FERREIRA and A. LEITÃO
Instituto Superior Técnico, Lisbon, Portugal
{sofia.feist, guilherme.barreto, bruno.b.ferreira}@tecnico.ulisboa.pt
antonio.menezes.leitao@ist.utl.pt

Abstract. Generative Design (GD) is a valuable asset for architecture because it provides opportunities for innovation and improvement in the design process. Despite its availability for Computer-Aided Design (CAD), there are few applications of GD within the Building Information Modelling (BIM) paradigm, and those that exist suffer from portability issues. A portable program is one that will not only work in the application it was originally written for, but also in others with equivalent results. This paper proposes a solution that explores portable GD in the context of BIM. We also propose a set of guidelines for a programming methodology for GD, adapted to the BIM paradigm. In the end, we evaluate our solution using a practical example.

Keywords. Building information modelling; generative design; portability; programming.

1. Introduction

Despite the fact that CAD tools support efficient working processes in architectural design, the creation of complex geometries can still be a challenge. This can be simplified with the use of GD, an algorithmic-based approach to design (Garber, 2014). GD also enables the automation of repetitive, time-consuming tasks, relieving architects from tedious and error-prone work.

To take advantage of GD, several tools, such as Grasshopper, Visual LISP and RhinoScript, emerged, allowing architects with basic programming skills to develop programs that generate models in CAD applications.

Recently, BIM tools have been replacing former CAD applications and, given the advantages of GD, it is tempting to extend it to BIM, introducing an algorithmic-based approach to Building Information Modelling.

The CAD and BIM paradigms are very different, and using the latter involves some significant shifts in design methodologies. For example, while CAD tools mostly deal with geometry, BIM tools produce digital representations of building components, containing both geometrical information and data attributes, such as material composition and cost (Eastman et al., 2008).

The building components have parametric and associative rules that dictate their behaviour, such as a door that can only exist hosted in a wall, which help ensure that the building components behave more like their real counterparts. These rules also facilitate the designer's job by propagating changes in the design.

For the GD field, the implication is that the differences between the CAD and BIM paradigms entail differences in the corresponding GD programming methodologies which, so far, have not been properly addressed.

1.1. MOTIVATION

The BIM paradigm, like the CAD one, can also benefit from GD and, indeed, tools like Dynamo and GenerativeComponents combine GD with BIM. These tools depend on Application Programming Interfaces (APIs) which provide a programmatic form of interaction with a BIM application. However, the use of these APIs requires extensive knowledge of textual programming languages, such as C# and C++, which few architects possess.

To overcome this, visual programming languages were developed, offering a more beginner-friendly alternative. Although very appealing for developing small programs, these languages become a barrier for complex programs, making them difficult to understand, use, and modify (Leitão and Santos, 2011).

An important feature that is missing from these programming tools is portability: the ability to execute the same program in different tools, producing equivalent results. One could hope that this lack of portability would not imply the lack of portability of the generated models and, in fact, there are solutions for transferring models between BIM applications, such as the Industry Foundation Classes (IFC). However, case studies show that IFC is not an entirely reliable method of transferring models between BIM tools (Golabchi and Kamat, 2013).

In this paper, we address GD in the context of the BIM paradigm, while taking into consideration the problems previously discussed. Our contributions are: (1) a solution for portable GD for BIM; (2) guidelines for a programming methodology for GD, adapted to the BIM paradigm. In the end, we evaluate our solution using a practical example.

2. Related work

There are several tools that address programming for BIM, including tools that are already available in BIM applications and plug-ins developed for the production of GD programs.

Grasshopper is a visual programming language for Rhino. Programs written in this language represent a data flow graph that consists of components and connections between them (Payne and Issa, 2009). Grasshopper can be extended with plug-ins for interoperating with BIM tools.

Lyrebird is a plug-in developed by LMN Architects to structure the information needed to identify and instantiate the correct BIM families in Revit (Logan, 2014).

Rhino-Grasshopper-ArchiCAD is another plug-in that allows the creation of BIM objects in ArchiCAD by using geometrical information created in Grasshopper (Graphisoft, 2015).

Dynamo is a plug-in for Revit that is strongly influenced by visual programming languages. Just as Grasshopper, users create a workflow by introducing nodes that are connected to each other through wires associated with the ports that each node contains (Autodesk, 2015).

GenerativeComponents (GC) is a parametric and associative system developed for Bentley's Microstation. GC has three ways of user interaction: (1) by direct manipulation of geometry; (2) by defining relationships among objects with simple scripts in GCScript; and (3) by writing programs in C#, allowing the definition of complex algorithms (Aish and Woodbury, 2005).

RevitPythonShell is a plug-in developed for Revit that allows users to take advantage of the RevitAPI but using Python. This tool was developed to simplify the workflow needed in order to use the RevitAPI (Thomas, 2009).

3. RosettaBIM

One flaw of the discussed tools is their lack of portability: a GD program written in one of those tools will only work with one specific BIM tool. In order to support the portability of GD programs, we propose RosettaBIM, a solution that considers BIM tools as back-ends for model generation.

RosettaBIM is composed of two components that exchange information through a communication channel. The first component is an abstraction layer that provides parameterised operations to create BIM objects, while the second is a plug-in that enables the production of the specified objects in a BIM tool. This architecture allows the creation of a common ground between the supported tools, enabling the portability of GD programs. Figure 1 shows a diagram of the proposed solution.

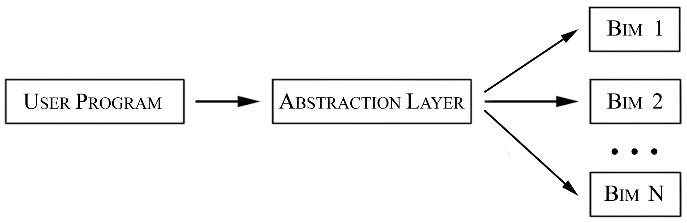

Figure 1. A simple diagram depicting the overall architecture of RosettaBIM.

In order to achieve the desired portability, the operations provided by the abstraction layer allow the creation of BIM objects in different back-ends. These operations are independent of the chosen back-end, which means that they have the same parameters. However, they are used differently according to the requirements of each plug-in. For example, to create a slab in a specific BIM tool, the operation might require a relative height while, in another tool, it might require an absolute height. To make this behaviour uniform, the operation in the abstraction layer requires the absolute height and the necessary conversions are made in the respective plug-in.

RosettaBIM is also designed to support the associative rules of the BIM paradigm, as these rules work both on the user's program and in the generated model. For example, to create a window in a BIM application it is necessary to have a wall as a host. This is reflected in RosettaBIM by forcing the operation that creates the window to also receive the wall as a parameter. This imposes the intended constraint in the creation of objects and in how the user organises his program.

In spite of the normalization done, we also want to enable the full exploration of a specific tool's capabilities. To this end, we grant the ability to relinquish portability in order to use those specific capabilities.

3.1. IMPLEMENTATION

In order to support several BIM back-ends we extended Rosetta (Lopes and Leitão, 2011), a tool that already provides portable GD for CAD. Although Rosetta did not support any BIM tools, it provided features that are useful in the BIM paradigm, such as coordinate systems and geometric abstractions. Moreover, Rosetta uses DrRacket (Findler et al., 1997), a pedagogical programming environment.

To support BIM tools, we established a communication channel to exchange information between the abstraction layer of Rosetta and the new BIM back-ends, and we extended that abstraction layer to provide new operations that support the BIM paradigm, particularly to create and manipulate BIM objects, including walls, doors, and stairs, among others.

Currently, this approach only supports Graphisoft's ArchiCAD and Autodesk's Revit as back-ends but can easily be extended to support other BIM tools. While all these tools share the same paradigm, there are substantial differences in some of the provided features. For example, Revit's API offers a more complex and detailed range of attributes for the creation of stair objects, while ArchiCAD's API allows the user to change attributes of slabs, like its thickness, without changing families.

Some of these differences were overcome through normalization while others were made available as BIM-specific features. In the next section, these trade-offs are further explained in the context of the case study.

4. Evaluation

To evaluate RosettaBIM, we selected an architectural case study: the Absolute Towers, designed by MAD Architects (Figure 2). In the next sections, we address GD for BIM, proposing guidelines for a BIM programming methodology, we analyse the portability of RosettaBIM, and we discuss the advantages of GD for BIM.

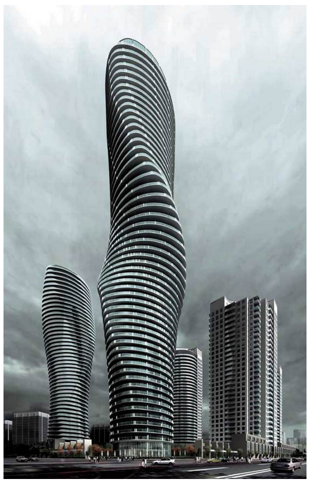

Figure 2. Absolute Towers, designed by MAD Architects and located in Mississauga, Canada (taken from Welch, 2014).

4.1. GENERATIVE DESIGN FOR BIM

In order to analyse GD programs for BIM, it is important to understand the main differences between working with CAD and BIM tools. Due to the differences in both paradigms, the corresponding GD programs will also differ. To understand these differences, we generated one of the Absolute Towers both with a GD program for CAD (henceforth, designated AT-CAD) and a GD program for BIM (henceforth, designated AT-BIM).

One major difference is that, contrary to a CAD tool, a BIM tool does not just create geometry; it creates digital representations of building components containing all the semantic information related to that component (AGC, 2005). As an example, consider generic slabs and walls: although they are geometrically similar, they are semantically different and, in a BIM tool, they must be created using different operations.

Fortunately, to increase the legibility of programs, good programming practices already promote the use of intermediate abstractions. For example, to make AT-CAD more legible, we implemented different user-defined functions for each building component, namely slabs and walls. These abstractions, although useful for organising the program, do not have any additional effect on the CAD tool besides the creation of the corresponding geometric objects. However, in the case of a GD program for BIM, the abstractions are, in fact, pre-defined operations and, thus, transfer the intended semantics to the generated objects. A positive side-effect is that, by dispensing the intermediate user-defined abstractions, AT-BIM becomes smaller than AT-CAD.

Pre-modelled building components are another difference between GD for CAD and GD for BIM that makes the creation of these components easier, since their geometry does not need to be created from scratch. For example, a door can be selected from a BIM library, while in CAD all of its subcomponents might need to be modelled. To overcome this disadvantage, CAD tools can import external blocks that are pre-modelled shapes or forms which can facilitate the design process of complex geometry. However, typically, these objects are not as parametric as the ones available for BIM tools, restricting the designer's ability to manipulate the geometry as they see fit.

Finally, BIM tools are more restrictive in the manipulation of the geometry of created objects and, as such, have several limitations regarding, e.g., Boolean operations. Nonetheless, we found such operations to be useful in the modelling of the Absolute Towers, namely in the adaptation of the walls to the rotation of the slabs which results in different wall lengths in every floor, as is visible in Figure 3. Using CAD tools, this could be achieved with an intersection between the walls and the volume corresponding to the interior of the building in each floor. Using BIM tools, the same had to be done using a different method: we implemented intersections between walls and slabs which compute the length of the walls in relation to the outline of the slab.

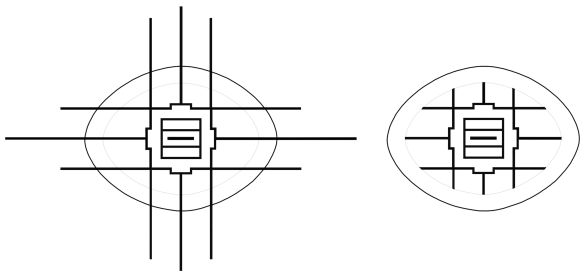

Figure 3. The length of the walls is computed by an intersection with the slab represented by a grey line.

4.2. PORTABILITY

In order to evaluate the portability of RosettaBIM, we tested it with a GD program that generates one of the Absolute Towers and we compared the generated models in the two supported BIM tools (Figure 4).

Given that the data from the used operations is manipulated by the plug-ins to meet the tool's requirements, we noted that the generated building components are created with the attributes that are relevant for each BIM tool. For example, whereas the slab created by RosettaBIM in Revit belongs to a Revit family and possesses its properties, the same slab created in ArchiCAD has properties according to the structure of the created slab.

Although we were able to normalize most operations, there are specific operations that are substantially different between Revit and ArchiCAD. For example, Revit allows the specification of an object's family, while ArchiCAD allows the specification of an object's properties. The corresponding BIM-specific implementation of these operations reflect these differences by accepting either a family, in the case of the Revit back-end, or the properties values, in the case of ArchiCAD. However, we also provide default values for all these parameters so that the generic use of the operation becomes identical in both back-ends, making the code portable and mitigating the differences between back-ends.

In order to further test RosettaBIM's portability, we compared it to another alternative based on the generation of the desired model in one BIM application and its transfer to another one via IFC. We tested the viability of this alternative in both directions: from ArchiCAD to Revit and conversely.

When transferring from ArchiCAD to Revit, although the geometry of the building remained the same, there were inconsistencies in the transferred building components. For example, the stair element, despite being named as such, did not have an associated Revit family.

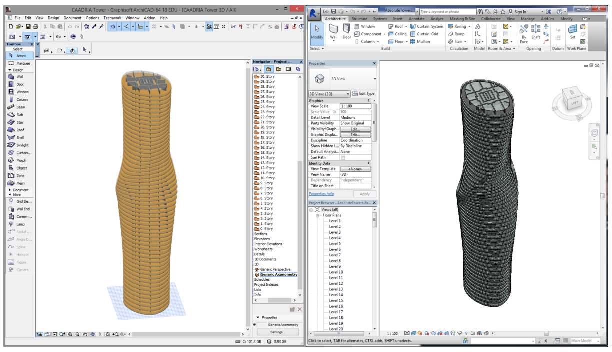

Figure 4. The Absolute Tower generated in ArchiCAD (on the left) and Revit (on the right).

Regarding the transfer from Revit to ArchiCAD, there were problems in the correct identification of some objects. Such was the case of roofs and stairs that lost their BIM properties, turning them into generic objects. In addition, the transferred stairs used a generic IFC material that did not match the original one.

4.3. ADVANTAGES OF GD FOR BIM

RosettaBIM provides GD for BIM applications, that is, an Algorithmic-based Building Information Modelling, which we name A-BIM. Given its potentially disruptive nature, it is important to identify the advantages that A-BIM can bring to the BIM paradigm.

A-BIM creates a paradigm shift in the design process, since the designer, instead of constructing the model directly in the BIM application, constructs the algorithm which generates the model in the BIM application.

Due to its algorithmic origin, the generated model is highly parametric, allowing us to easily experiment with different values of parameters, producing a wide range of possibilities that can be explored and evaluated. This is visible in Figure 5, where we explore different shapes for the tower's slabs.

In addition, because the generated objects are parametrically interdependent, changes are propagated to the entire model. Although this feature is not exclusive to A-BIM, it is more flexible than the one typically available in BIM tools. For instance, whilst in manual BIM a change in the torsion angle of the tower requires the manual update of all building components, in A-BIM it is possible to automatically propagate that change to all building components, with the result visible in Figure 5.

Although our approach allows the exploration of parametric models, it requires programming knowledge and an initial investment to produce the algorithm, which might take more time than producing the model manually.

On the other hand, this initial effort can be rewarded in projects where a high level of exploration is required or in highly parametric buildings.

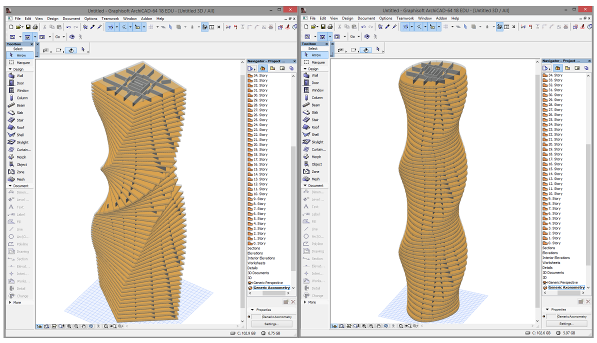

Figure 5. Absolute Tower variations. On the left: different slab shapes. On the right: different torsion angle.

5. Conclusion

RosettaBIM implements portable GD for BIM. This is achieved by an abstraction layer that provides normalized operations for every supported BIM application. However, when necessary, the user can relinquish this portability in favour of accessing tool-specific features.

In order to use RosettaBIM, users need programming skills and an initial time investment for the development of the GD program. This might be seen as a barrier, but the rewards of this investment become clear as the project grows and reaches a higher complexity. This is noticeable in projects where design exploration and frequent changes are a requirement.

RosettaBIM, by supporting GD for BIM, introduces changes in the way GD programs are written compared to how they are written in GD for CAD. Given that the modelling operations in BIM have semantics and introduce dependencies between objects, this requires the user to organise his program in a way that respect those dependencies.

In this paper, we demonstrated that RosettaBIM supports the development of portable GD in the BIM context by showing an algorithmic version of the Absolute Towers that creates equivalent models in two different BIM applications. Also, every generated building component has the corresponding properties that are appropriate for each BIM application.

Furthermore, we showed that the commonly accepted alternative for portability – the IFC data model – still has problems, such as loss of information. Be that as it may, IFC would not be the ideal solution for porting models generated by programs, since any modification would require the generation of the model in one tool just to transfer it to the other one.

With RosettaBIM we can explore a methodology for GD in the BIM paradigm that we name A-BIM, acronym for Algorithmic-based Building Information Modelling. Using this methodology, we could achieve a higher degree of flexibility than what is possible with the manual approach.

In the future, we will expand RosettaBIM to support more BIM tools as back-ends, such as Digital Project or Microstation; and add more modelling operators and constraints, allowing the generation of more complex models.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

Aish, R. and Woodbury, R.: 2005, Multi-level interaction in parametric design, in Butz, A. et. al. (eds.), *Smart graphics*, Springer Berlin Heidelberg, Berlin, 151–162.

Associated General Contractors of America: 2005, The contractor's guide to BIM, *AGC Research Foundation*, Las Vegas.

Autodesk: 2015, *"The Dynamo Primer"*. Available from: Open Source Repository <http://www.dynamoprimer.com/> (accessed 25 November 2015).

Eastman, C., Teicholz, P., Sacks, R., and Liston, K.: 2008, *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers, and contractors*, John Wiley & Sons, Inc., New Jersey.

Findler R., Flanagan C., Flatt M., Krishnamurthi S. and Felleisen M.: 1997, DrScheme: A pedagogic programming environment for Scheme, *in* Glaser, H. et al (eds.), *Programming languages: implementations, logics, and programs*, Springer Berlin Heidelberg, Berlin, 369–388.

Golabchi, A. and Kamat, V.: 2013, Evaluation of industry foundation classes for practical building information modelling interoperability, *Proceedings of the 30th ISARC*, Montréal, 17–26.

Graphisoft: 2015, *"GRAPHISOFT introduces Rhino–Grasshopper–ARCHICAD connection"*. Available from: Open Source Repository <http://www.graphisoft.com/info/news/press_releases/graphisoft-introduces-rhino-grasshopper-archicad-connection.html> (accessed 20 November 2015).

Leitão, A. and Santos, L.: 2011, Programming languages for generative design, *eCAADe29*, Ljubljana, 549–557.

Logan, T.: 2014, *"Superb Lyrebird"*. Available from: Open Source Repository <http://lmnarchitects.com/tech-studio/bim/superb-lyrebird/> (accessed 25 November 2015).

Lopes, J. and Leitão, A.: 2011, Portable generative design for CAD applications, *ACADIA 11*, Banff, 196–203.

Payne, A. and Issa, R.: 2009, *The grasshopper primer*, Robert McNeel & Associates.

Thomas, D.: 2009, *"Introducing RevitPythonShell"*. Available from: Open Source Repository <http://www.darenatwork.blogspot.pt/2009/12/introducing-revitpythonshell.html> (accessed 25 November 2015).

Welch, A.: 2014, *"Absolute Towers, Canada : Mississauga Architecture"*. Available from: Open Source Repository <http://www.e-architect.co.uk/canada/absolute-towers-mississauga/> (accessed 25 November 2015).