ANALYTICAL MODELLING IN DYNAMO

Marek SALAMAK, Marcin JASINSKI, Tomasz PLASZCZYK, Mateusz ZARSKI

Department of Mechanics and Bridges, Faculty of Civil Engineering, Silesian University of Technology, Akademicka 5, Gliwice, Poland

marek.salamak@polsl.pl, marcin.jasinski@polsl.pl, tomasz.plaszczyk@polsl.pl, mateusz.zarski@polsl.pl

DOI: 10.31490/tces-2018-0014

Abstract. *BIM is applied as modern database for civil engineering. Its recent development allows to preserve both structure geometrical and analytical information. The analytical model described in the paper is derived directly from BIM model of a structure automatically but in most cases it requires manual improvements before being sent to FEM software. Dynamo visual programming language was used to handle the analytical data. Authors developed a program which corrects faulty analytical model obtained from BIM geometry, thus providing better automation for preparing FEM model. Program logic is explained and test cases shown.*

Keywords

Analytical model, automation, BIM, Dynamo, Revit, visual programming, VPL.

1. Introduction

BIM (Building Information Modelling) is a modern technology, slowly replacing current CAD (Computer Aided Design) software. Its aim is to replace scattered building information stored within different files with one unified model containing all building data. Current solutions usually divide information base on its type. Usual division is into textual data (documentation) and graphical data (geometry, drawings). Documents are stored separately from structure drawings. Even inside their genre – data is usually unrelated. Drawings are not linked and their information is modelled with simple objects such as text, lines, arcs etc. BIM however, contains solution to most of those problems. Both documentation and drawings are contained within single model. The first one is attached as additional information to the objects, second ones are generated on the fly from 3D model. Model objects are divided into classes and modelled as whole entities such as beams, walls, slabs etc. They contain information not only about geometry, but also about features normally contained in the documentation such as: material type, analytical properties, phase, cost etc. Model elements have defined relationships between them. In such a way model becomes a relational database. Categorized data can be then queried, searched and managed with appropriate software. Adoption of BIM results not only in utilizing single unified model but also in adopting relevant building documentation process. It requires constant data exchange between all members. More demanding at design phase – it provides savings in building management phase.

Structure analytical information is today mostly kept within FEM models. It contain basic information about analytical geometry and calculation results. Model data is stored separately from rest of documentation. BIM analytical modelling tools approaches this issue, however without sufficient results. Models generated automatically are still required to be manually adjusted. Methods of automating that process are being searched. This problem could be solved with different programming techniques, beginning with rule-based systems, finishing with AI (Artificial Intelligence) Deep Neural Networks. This article as a first of series provides results of rule-based approach.

2. Background

Analytical modelling in BIM has been introduced with Autodesk Revit environment. Well documented API (Application Programming Interface) allows for seamless integration with other software. Dynamo is extension software operated through Revit. It is an interface between visual programming environment and Revit API.

Autodesk Revit contains both physical (geometry) and analytical model. Analytical model is connected to its physical representation. Change in physical model results in analytical model change. This is a one way relationship, which can be disconnected. Physical representation contains standard geometry and BIM data. Analytical consists of separate, simplified 3D element representation.

It also contains material properties of an object, boundary conditions, and applied loads. Automatically generated analytical model is usually imperfect and has to be improved manually.

Visual Programming Languages, in short VPL, are 2D representation of standard textual programming languages. Their usage is growing in recent years, mainly due to higher computational capacity of hardware. Mayers in his research [1] concludes that contrary to textual languages, they have more steep learning curve which makes them easier to master in relatively shorter time. Textual languages however, are significantly more developed. They possess vast documentation and very good libraries. Their execution speed is faster and memory footprint smaller. Code is clearer and its management easier. For the presented rule-base system approach, visual programming language was used.

Execution flow is a description of program execution sequence. There are two main types: control and data flow. First is determined by specific instructions (conditionals, loops etc.), second with data passing. Use of latter is more risky due to certain unpredictability of element execution time. It's difficult to predict which code fragment will be calculated faster and pass data further. On the advantages however, it is easier to implement it in visual programming languages.

To describe classification of programming languages, programming paradigms are used. They define the style of the code. Main paradigms used both in textual and visual programming languages are:

- procedural/structural – mainly recognized in base programming languages such as Pascal and C. Flow is controlled with loops (while, for) and conditions (if, case, switch),
- functional – pure functional languages are rare and usually applied in mathematics. Examples are LISP, Schema. Flow control is managed by function call order. Biggest advantage of functional languages is ease of use of recurrence,
- object-oriented – most recent paradigm, used in languages such as Java, C++ and C#. It brings data and functions together by creating an object. It introduces mechanism such as abstraction of data (hiding data), encapsulation (binding data with functions), inheritance (acquiring properties of another object) and polymorphism (functions prepared for different data types). Very flexible, designed for producing complex software.

Dynamo is both a visual programming language and integrated development environment (IDE). User interface consists of two separate workspaces with program code and background geometry preview (see Fig. 1). By default program is interpreted, allowing for code adjustments in a real time. Code is presented as nodes connected with wires. Program flow is controlled by data. Calculations are done in node that currently received all data from its input. Dynamo also supports functional programming paradigm. Part of nodes are containing data while other ones – functions. Unfortunately, support for procedural programming paradigm is missing. It reveals itself in lack of loop (for, while) instructions. Latest can be obtained only with use of recurrence (nesting functions inside each other). Direct support for object-oriented programming is also missing. Those features can be however alleviated with use of custom nodes. Dynamo allows for creation of own specific nodes – using textual languages such as Python and C#.

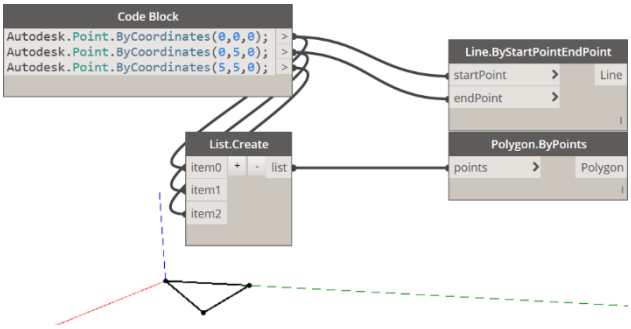

Fig. 1: Dynamo program code and geometry preview.

Analytical Modelling for Dynamo was proposed as a rule-based system solution to solve analytical model inconsistency problems. Project was designed to utilize as most of Dynamo own internal functions as possible. However, if missing, development was done using Python and C# languages. Prepared algorithm is recalculating and correcting analytical model using parameters obtained from user. Most important parameters, are: default (reset) structural model switch, inconsistency detection tolerance, structural elements connection priority list and elements adjustment switch (see Fig. 2).

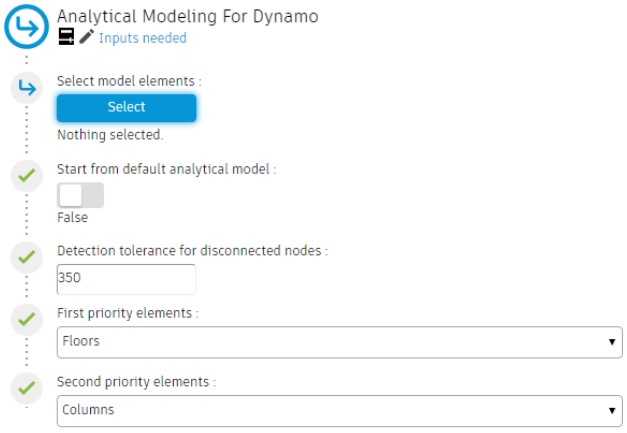

Fig. 2: Part of Dynamo program interface.

User specified tolerance is used to align structural elements in order provided with connection priority list. It is used to specify which elements should be aligned and which should remain unchanged. Program can be run multiple times to achieve desired effect. Elements can be also adjusted within its own categories (e.g. disjointed connection between beams or slabs). If user is not satisfied with obtained result it is possible to restore it to the default location with reset switch prior the analysis. Detailed description of this process is provided (see e.g. Sec. 3).

3. Numerical modelling in BIM

3.1. Overview

In the considered BIM modelling software both models, physical and the one for numerical analysis, are created simultaneously. By Revit glossary, the numerical model is usually called the analytical model [2].

A physical model is a 3D visualization of a building. Each element of the model is additionally described using collections of parameters and building datasets. Accuracy of the physical model is usually greater than accuracy of the analytical model. The physical one consists of spatial instances and solids, including non-structural elements that are eventually ignored during numerical analysis. Physical parts build a core of the building database and are crucial in the BIM-oriented process hence the physical model is used to be called directly a BIM model. Data introduced to the BIM model are complemented during the whole life cycle of a building and can be received from the model at any time [8]. Such the approach replaces extensive repositories of files and usual paper archives. All of the documents are produced during stages: conceptual, design, construction and operational. Putting them all together in traditional volumes is usually characterized by significant dispersion, inconsistency and limited usability at future stages. Transferring a building database to digital environment built upon a BIM model allows to rearrange the whole base and make the information accessible and usable for information systems used in the building process.

The idea of BIM is based on communication between several interlinked IT systems that send data to the model, receive it when requested and reprocess it on their own. This approach was used in cost estimating procedures that use the model as a base of quantitative data [3]. The building model was also introduced in scheduling and monitoring of ongoing construction by comparing with real progress on site. It was possible to achieve by applying point cloud data to the model [4] or pairing the model elements with RFID tags [5]. BIM models were used in optimization of schedules with clash detection performed at the same time [6]. Checked methods include genetic algorithms [7] [9], simulations [10] or spatial reasoning [11]. Known attempts to connect the post-construction stages with a BIM model are based on laser scanning, photogrammetry and similar. In [12] it was proposed to evaluate quality of construction works by comparing basic model with a view captured by such the digital techniques. Authors of [13] suggest calculating performance indices by detection of damages using a laser scanner and introducing them to BIM models. Inspection of bridges in the context of Industry 4.0 trends was described in [14].

In the examined BIM environment, an analytical model is a resultant of a physical model. An exemplary analytical model of a structure was shown in the Fig. 3(a) and its associated analytical model – in the Fig. 3(b). Each of the structural elements of the model can be described by properties defining its role in the analytical model. In this way it can be determined if the element should be part of the structure that bears loads or, on the contrary, its role is secondary as in the case of finishing elements, e.g. partition walls or pilasters. Such the information is integral part of the building database.

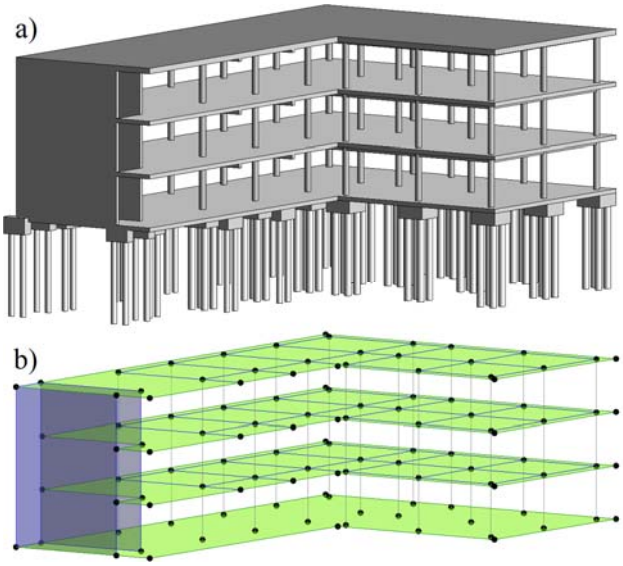

Fig. 3: Physical BIM model of a building (a) and associated analytical model (b).

In a BIM model, elements also contain information regarding their function in the object. Basic structural components can be distinguished: beams and bracing, columns, slabs and walls. Physical instance of a beam is associated with a congruent analytical bar. Similar information is assigned to bracing and columns as linear elements. Planar structures, such as slabs and walls, generate analytical shells that are associated with their boundaries and placement. Changes that were introduced to the physical model are automatically reflected in the analytical model. It is also possible to introduce changes to the analytical model, however, it usually does not lead to appropriate changes in the parent physical model. The analytical model created and modified in this way is a subset of the BIM model. Such the subset is then exported to FEM software where numerical analysis is performed.

3.2. Limitations

Automatic process of generation of an analytical model, parallel to creation of a physical model, is based on internal rules and algorithms of the software. They form a part of the software structure and are not accessible for a user. For example, a default axle of an analytical column passes through the reference point defined by the user during defining the cross-section shape of the physical solid. By contrast, a default axle of an analytical beam is projected at the top surface of the physical instance of the beam. Wherever it is possible, the algorithms included in the software look for inconsistencies between joined elements. The inconsistences that were found are automatically corrected, e.g. by elongating, shortening, projecting and/or

moving vertices of an analytical column to face of an analytical shell found nearby the defective area. The range of such the correction is controlled by global variables of the project. The user is able to set a numerical value of a maximum allowable translation of an analytical vertex from its original point in horizontal and vertical directions. In Revit, the value is set to 12 inches by default. If the correction requires greater distance to be fixed, it is not performed. Beside the automatic correction of the analytical model, the user is able to control its adjustment by manual modification. In practical terms this involves moving vertices of analytical bars or shells to new positions indicated by movement of a computer mouse. One of the analytical element properties is projection that parametrically define location of the analytical element in relation to its physical parent. For example, an analytical beam can be translated to the top (default) face of the physical solid, to half the height of the solid or its bottom face.

The automatic correction implemented by software developers does not always lead to full consistency of the analytical model. Such the undesirable results may be caused by general complexity of the internal algorithm that is not able to detect and resolve all possible connections between elements correctly as well as by risk of erroneous process of creating physical parts of the model by the user. The manual correction at the disposal of the user can be carried out with only a limited accuracy. In the case of huge and complex models, the manual correction can be time-consuming and ineffective. If the inconsistencies are left without intervention of the user and transferred directly to the numerical solver, results of the numerical analysis are incorrect.

Due to described difficulties, an attempt was made to create an algorithm that would be responsible for additional analysis and correction of the analytical model. It is based on separated set of rules and works as an add-in to the Revit software. The algorithm was designed in Dynamo visual programming language with integrated Python scripts and individually designed C# classes.

3.3. Analysis and dataflow

Proposed algorithm runs on set of input data, including: selection of objects to analyse, tolerance, priorities of element types (categories) and two Boolean switches – enabling or disabling model reset prior to analysis and improving convergence between elements of the same category.

As part of the selection process, the whole model or its part is selected. The selected objects will be modified by the algorithm meanwhile unselected ones will be ignored. From the set of selected elements, only physical objects that are structural parts of the model and own their analytical models are passed through to further analysis. As a result of internal filtering, all non-structural objects (e.g. finishing elements, equipment, doors, windows, networks or partition walls) are omitted as well as the structural elements that were set by the user to be ignored in analytical model. Setting an element to not generate analytical model is the internal part of Revit environment and is used regardless of the proposed algorithm.

The tolerance is a numerical value defining range and interactions between analytical elements. If a distance between two disconnected nodes (vertices) of two different types of objects is less or equal the tolerance, the algorithm will attempt to translate one of the nodes to the second one, depending on the defined priority chain. The purpose of the tolerance parameter is similar to the one in automatic correction implemented in Revit software. The range of influence of a node on the rest of nearby nodes can be visualized by the tolerance bubble – a ball pinned in the given node (see Fig. 7).

The priority chain describes relations between basic types of structural elements. Four basic types are established: framing (general term for both beams and bracing), columns, slabs and walls. Analytical elements assigned to them are: analytical framing, analytical columns, analytical slabs and analytical walls respectively. Taking into account that every type of elements can interact with other types and elements of the same type, 16 relations should be considered. Still it should be noted that relation A-B is not equal to relation B-A, where letters A and B mean any different types of elements.

Figure 4 shows a simple model consisting of a single column and a slab. In the example it can be expected that the vertex of the slab will be translated to the column axle or, alternatively, top end of the column will be moved to the corner of the slab. Hence, two antagonistic relations are to be considered: column – slab and slab – column respectively. When calling relations, following assumption was made: the first field in the name of relation means the master element and the second one means the slave – the object that should be snapped to the master element.

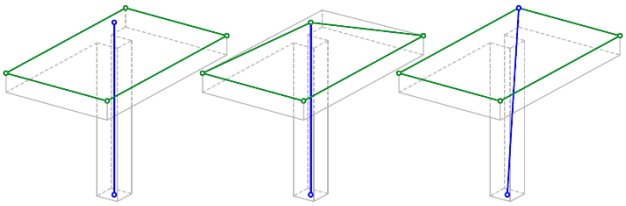

Fig. 4: Exemplary analytical model derived from Revit physical model: original (on the left), modified by relation column – slab (in the middle) and modified by relation slab – column (on the right).

By setting priorities, the user decides which possibility is more accurate in his case. The rest of 14 relations between different types of elements may occur in other, more advanced and complex models. In general, it is possible to define a three-level priority chain by assigning: the type of elements of the highest priority, the type of elements of middle priority and the type of elements of the lowest priority. The analytical element that belongs to the type of the highest priority will not be modified. However it will snap two types of lower priorities to itself. The analytical element of middle priority will snap the elements of the lowest priority only. The order of resolving the priority chain was shown in the Fig. 5.

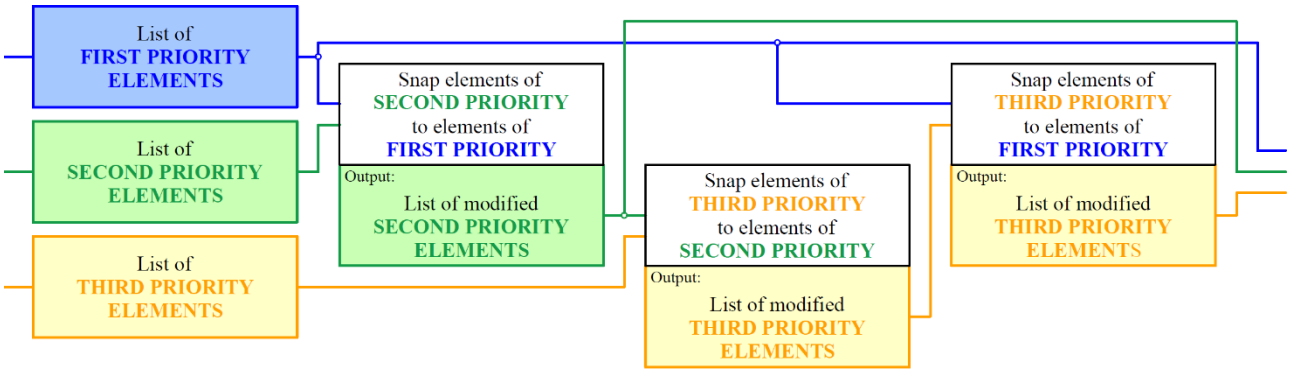

Fig. 5: Schema of modification path for elements grouped by priorities assigned to three of four available element categories.

For each of the relations, dedicated sub-algorithm was designed. Each of the sub-algorithms is responsible for detection of disconnections between two desired types of elements, resolving translation vectors for vertices of the elements of modifiable type and forcing the translations. Such the 16 sub-algorithms are parts of the network of the whole script. Exemplary operating scheme of a sub-algorithm was presented below by the case of two selected antagonistic relations: column – slab responsible for aligning slabs to columns and the opposite one: slab – column responsible for aligning columns to slabs. The order of operations will be presented using the same part of the analytical model shown in the Fig. 4.

At this stage the internal sub-algorithm receives two lists of elements. One of them consists of elements of the higher priority. The second one consists of elements of the lower priority that will be finally modified by the sub-algorithm. These types will be called master and slave respectively. On the assumption that the listed elements are physical objects, the first step to process by the sub-algorithm is extracting analytical elements from their physical parents. The analytical elements are then converted into Dynamo geometry. Prior to the conversion the elements are described by a label and a unique index assigned by Revit. After the conversion they become a set of mathematical objects defined by processable and calculable topology including faces, edges and vertices. These are then processed and changed for surfaces, curves and points respectively. Further operations are carried out on the lists of geometry.

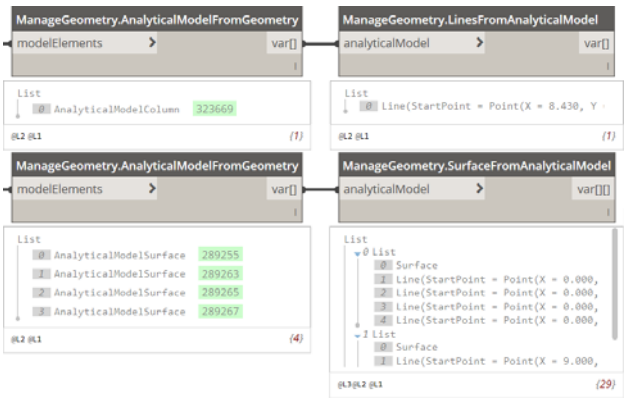

Fig. 6: Dynamo code: analytical instances passed from physical objects, passed to nodes responsible for converting them into geometry.

A translation vector for a vertex of an analytical element is defined by two points: starting point and end point. The starting point is the given vertex of the slave element before the translation. In the described example, in the case of relation responsible for modification of columns, the starting points are both ends of the analytical column. In the case of relation modifying slabs, the starting points are all vertices of all corners of the analytical slab. After the conversion into Dynamo geometry, their coordinates are obtained.

The end point of the vector is always the point lying on the master element. Obtaining the point is based on intersections of the geometry after conversion of master elements with a tolerance bubble. The tolerance bubble is a ball of diameter equal to the tolerance, pinned at the starting point of the sought vector. The tolerance bubbles for both of exemplary relations are shown in the Fig. 7.

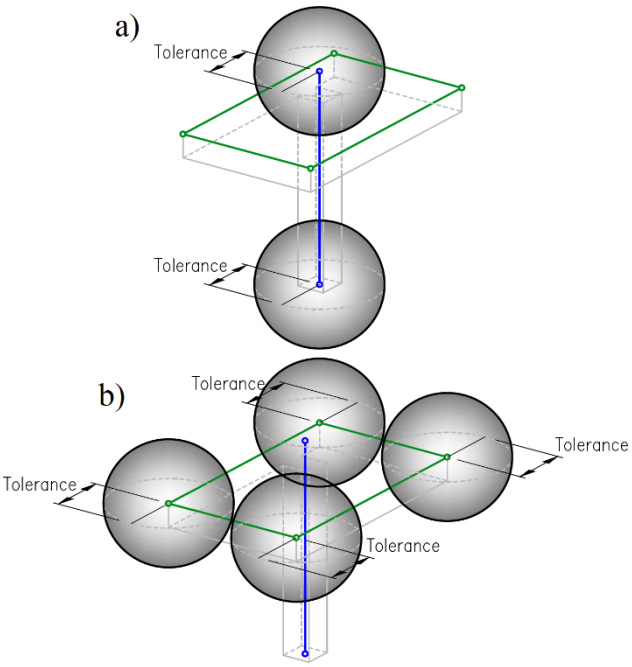

Fig. 7: Tolerance bubbles pinned at vector starting points: relation slab – column (a) and column – slab (b).

The intersections are sets of common parts: of the ball and geometry obtained from nearby master elements. Intersections are derived from operation performed using

one of the Dynamo nodes called *Geometry.Intersect*. Input data are: list of geometry obtained from the conversion of analytical elements and list of the tolerance bubbles. Lists passed to the *Geometry.Intersect* node are crossed to seek for intersections of each geometrical object with each of the tolerance bubbles. Products of intersections for the exemplary relations are shown in the Fig. 8.

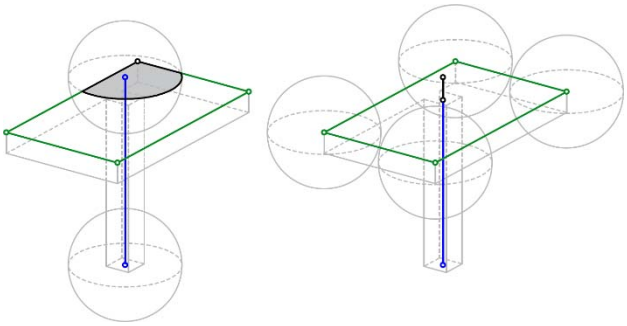

Fig. 8: Intersection products for both exemplary relations.

Further, using the *Geometry.ClosestTo* function node, coordinates or the closest point lying on the intersection products from the center of the ball are obtained. In the given example, in the case of relation modifying analytical columns, the balls are pinned at both ends of the column. Intersection products are found only for the top ball. These include part of the surface of the analytical slab and boundary objects: two edge curves and the vertex in the slab corner. In this kind of relation, the sub-algorithm checks if the intersection products contain vertices. If so, the closest vertex is the end point of the vector. If no vertex is in the set of intersection products, the sub-algorithm checks if the products contain edges. If so, the *Geometry.ClosestTo* node obtains the point that lays on the edges and is the closest one to the start point. If the intersected part of the analytical slab does not contain vertices, nor edges, the *Geometry.ClosestTo* node attempts to find the closest point on the surface. Finally the end point of the vector is obtained. In the case where intersecting node does not find any intersection between master elements and the tolerance bubble (e.g. the base of the column in Fig. 8), the list of intersection products is empty and the translation vector is a null vector.

Similarly in the example of relation modifying slab in relation to columns. The balls are pinned in each of the slab corners. For one of the corners products of intersection with the analytical column were found. For the rest of three corners null vectors are returned. The *Geometry.ClosestTo* node obtains the closest point on the part of the analytical column derived from intersection with the tolerance bubble. Finally the point is the new vertex of the slab boundary that will replace the original vertex in the corner.

Translation vectors are arranged in lists. In the case of relation modifying columns, each of the analytical columns receives two vectors – one vector for the base of the analytical column and the other one for the top of the column. The lists of vectors together with lists of elements IDs are passed to the authorship Dynamo nodes forcing appropriate translations: *SetStartOffset* and *SetEndOffset*.

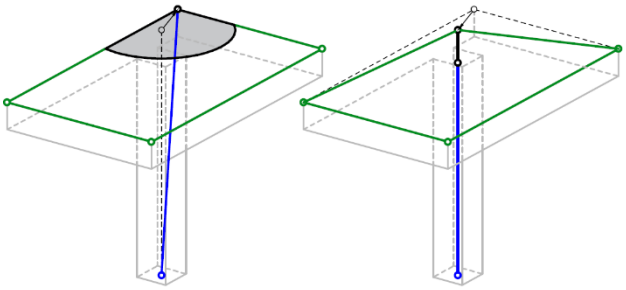

Fig. 9: Translation vectors for both exemplary relations.

The same nodes are used to translate vertices of beams and bracing. In the case of slabs, number of vertices is variable. Hence, apart from the list of IDs of elements, the authorship modifying node – *SetFloorBoundary* – takes the list of new boundaries. The new boundary is created basing on the original boundary where some of the original vertices are replaced for the new ones obtained from intersection products.

4. Test cases

4.1. Initial test cases

As it was stated before, the aim of the project was to develop a working algorithm that could handle repairing of complex analytical models built with Revit software, so in order to check if introduced operations are performed correctly, premade input models were needed.

The first cases were developed by the team not strictly in order to test the software, but to acquire knowledge on approximate number of ways of how and why the element joints can be disconnected in Revit environment. Tests resulted in elaboration of the total number of 93 possible ways of disconnection for all of the elements with analytical models featured in Revit software, from which the most common elements joints were connections between two linear objects, such as beams or columns (36 out of total number of 93 cases).

With further development of the script, the dictionary of connections became the initial collection of test cases for testing newly developed sub-algorithms described in sec. 3.3. It was later replaced by other models developed specifically to test single relation (single-case scenarios) and then again by models representing real construction designs, obtained from practical Revit applications.

4.2. Internal and external testing

For every newly developed piece of algorithm, both for handling different element categories as well as a single operation (e.g. translating elements), a specific sequence of following actions was applied regarding testing process.

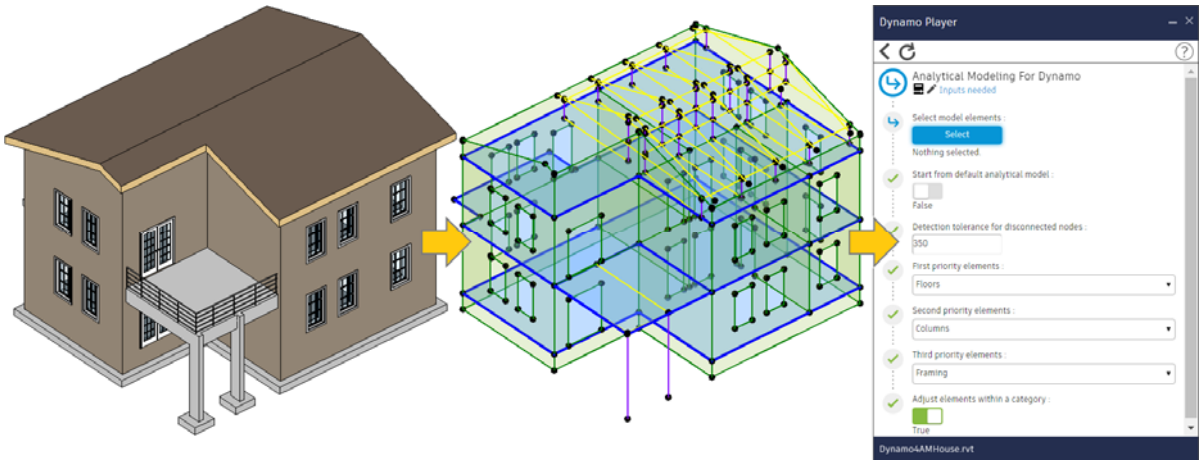

Fig. 10: An example of testing workflow.

Initially, every current algorithm addition was tested with a pair of models: the above-mentioned connection dictionary and a specific model built specially for one operation of the solution, called a single test case. If the initial tests proved algorithm to be working as intended, the second part of testing would be implemented. In the second part, testing was carried out on comprehensive, practical models brought up in the preceding paragraph. Those models contained parts or entire real constructions that had some imperfections in their analytical models which rendered them useless in numerical calculations (e.g. disconnected nodes, interpenetrating analytical slabs and similar). During the process of testing, the model geometry was updated given number of times with various initial settings of algorithm, as shown in the Fig. 10. The correctness of the outcome was checked after each run and compared with other models. Output logs were also analysed, using log4net logging interface. If the conducted tests proved the correct operation of the algorithm, they were considered as successfully completed and the tested functionality was then added to the next release of the open beta version.

External testing was conducted in a form of beta user acceptance testing, in which only release-ready beta versions of the solution were provided for testing. Those beta versions were delivered to three third party designing companies and then the collected feedback was considered during development of the next releases.

There are many arguments for this approach, but there were only few arguments prevailing on the choice of this method. The key arguments are: reduced testing time, the possibility to focus more on actual development rather than testing and independence from the developing team in validation.

The testing process in the beginning of the project tended to be somewhat chaotic and not focused on single tasks at hand, what often led to delays in algorithm development or minor errors in its operation in final releases. Since then, the whole process greatly improved, by utilizing the key principles in software testing and outsourcing some of tests to beta user testing. With this changes, the time needed to release next major update to algorithm could be reduced from around two months in the beginning of the project to less than month by its final date.

5. Conclusion

Authors presented introduction to the concept of data storage in BIM models. Existence of both physical and analytical model was explained. Shortcomings of that technology were pointed out. Biggest one is inconsistency of automatically created analytical model. Certain elements might be created displaced and disjointed, thus rendering model ineffective for future FEM use. Rule-based system was proposed to fix obtained model. Software implementation was done in Dynamo visual programming interface for Autodesk Revit. Proposed algorithm is based on tolerance and elements connection priority. User is requested to input the radius of sphere created at the control points of each element – within which detected intersections will be connected. Decision if to connect certain elements is made on base of priority list. Elements with lower priority will be adjusted to elements with higher one (e.g. beams aligned to columns). In the future research, it is proposed to introduce automatic, local tolerances – specific for certain elements categories. Tolerances could be adjusted using statistical information about distances within model. Element priority list automation problem will be also approached. As an example - it can be generated on the base of relational position between elements (e.g. elements placed lower in structure will remained unchanged).

Program was tested both by internal team and external partners. Internal testing has greatly contribute to the final concept of software. It influenced chain structure of algorithm and its division into sub-algorithms. External tests provided vast amount of exceptions which had to be dealt with. User feedback was necessary to predict situations which well very unlikely to happen under standard development testing procedure. After many versions of released software, most of tests came out positively. Next step will be releasing prepared software

for testing in construction industry. Further automation of flow between BIM and FEM is desired. Updated approach will have to incorporate most recent computer science techniques such as AI deep neural networks. With use of those methods, it will be possible to achieve almost seamless transition between BIM and FEM models.

References

[1] MAYERS, B. A., Visual programming, programming by example, and program visualisation: a taxonomy. In: *SIGCHI Conference on Human Factors in Computing Systems*. New York: ACM, 1986, pp. 59 – 66. ISBN 0-89791-180-6.

[2] CHARPENTIER, J., Deep Dive Into Revit® Structure Analytical Tools. In: *Autodesk University*. Las Vegas, 2007.

[3] SULBARAN, T., STRELZOFF, A., Conceptual processes for integration of a BIM software and a cost estimating software. In: *International Conference on Software Engineering Theory and Practice*. Orlando: ISRST, 2009, pp. 175 – 181. ISBN 978-1-61567-659-0.

[4] BRAUN, A., BORRMANN, A., TUTTAS, S., STILLA, U., Towards automated construction progress monitoring using BIM-based point cloud. In: *10th European Conference on Product and Process Modelling*. Vienna: CRC Press, 2014, pp. 101 – 107. ISBN 978-1-13802-710-7.

[5] CHEN, K., LU, W., PENG, Y., ROWLINSON, S. HUANG, G. Q., Bridging BIM and building: from a literature review to an integrated conceptual framework. *International Journal of Project Management*. 2015, vol. 33, iss. 6, pp. 1405 – 1416. ISSN 0263-7863. DOI: 10.1016/j.ijproman.2015.03.006.

[6] CHEN, Y. J., FENG, C. W., WANG, Y. R. WU, H. M., Using BIM model and genetic algorithms to optimize the crew assignment for construction project planning. *International Journal of Technology*. 2011, vol. 2, no. 3, pp. 179 – 188. ISSN 2086-9614. DOI: 10.14716/ijtech.v2i3.68.

[7] FAGHIHI, V., REINSCHMIDT, K. F., KANG, J. H., Construction scheduling using genetic algorithm based on building information model. *Expert Systems with Applications*. 2014, vol. 41, iss. 16, pp. 7565 – 7578. ISSN 0957-4174. DOI: 10.1016/j.eswa.2014.05.047.

[8] KUDA, F., WERNEROVA, E. & ENDEL, S., Information transfer between project stages in the life cycle of a building, *Vytapeni, Vetrani, Instalace*, vol. 25, no. 3, 2016, pp. 156-159.

[9] MOON, H., KIM, H., KIM, C., KANG, L., Development of a schedule-workspace interference management system simultaneously considering the overlap level of parallel schedules and workspaces. *Automation in Construction*. 2014, vol. 39, pp. 93 – 105. ISSN 0926-5805. DOI: 10.1016/j.autcon.2013.06.001.

[10] MARX, A. KÖNIG, M., Modeling and simulating spatial requirements of construction activities. In: *43rd Winter Simulation Conference – Simulation: Making Decisions in a Complex World*. Washington: IEEE Computer Society, 2013, pp. 3294–3305. ISBN 978-1-4799-2077-8.

[11] WELDU, Y. W., KNAPP, G. M., Automated generation of 4D building information models through spatial reasoning. In: *Construction Research Congress: Construction Challenges in a Flat World*. West Lafayette: ASCE, 2012, pp. 612–621. ISBN 978-0-78441-232-9.

[12] AKINCI, B., BOUKAMP, F., GORDON, C., HUBER, D., LYONS, C., PARK, K., A formalism for utilization of sensor systems and integrated project models for active construction quality control. *Automation in Construction*. 2006, vol. 15, iss. 2, pp. 124 – 138. ISSN 0926-5805.

[13] SACKS, R., AKEDAR, A., BORRMANN, A., MA, L., SINGER, D. KATTEL, U., See Bridge Information Delivery Manual (IDM) for next generation bridge inspection. In: *33rd International Symposium on Automation and Robotics in Construction*. Auburn: IAARC, 2016, pp. 728 – 735. ISBN 978-1-5108-2992-3.

[14] SALAMAK, M., JANUSZKA, M., BrIM bridge inspections in the context of Industry 4.0 trends. In: *Maintenance, Safety, Risk Management and Life-Cycle Performance of Bridges*. London: Taylor & Francis Group, 2018, ISBN 978-1-138-73045-8.

About Authors

Marek SALAMAK was born in Sanok, Poland. Assoc. Prof. at Silesian University of Technology in 1990. His research interests include bridges, BIM and infrastructure asset management.

Marcin JASINSKI was born in Katowice, Poland. He received his M.Sc. from Silesian University of Technology in 2015. His research interests include bridges, BIM, optimisation of processes and automation in construction.

Tomasz PLASZCZYK was born in Knurów, Poland. He received his M.Sc. from Silesian University of Technology in 2013. His research interests include bridges, dynamics, BIM, FEM and machine learning.

Mateusz ZARSKI was born in Będzin, Poland. He received his M.Sc. from Silesian University of Technology in 2017. His research interests include SHM systems, artificial intelligence and life cycle of structures.