Automatyzacja prac projektowych przy użyciu programu Panel Connector

Dariusz Rocki
ARUP Polska, ul. Inflancka 4, 00-189 Warszawa

Streszczenie: W pracy wskazano sposoby ułatwiające wdrażanie technologii modelowania informacji o budynku BIM w biurach projektowych przez wprowadzenie narzędzi optymalizacyjnych. Niniejsze opracowanie prezentuje propozycję oryginalnego podejścia do tego zagadnienia zrealizowanego w środowisku Dynamo z użyciem języka Python i dokumentacji Revit API, które umożliwia automatyczne podłączanie do wybranej tablicy opraw oświetleniowych, gniazd elektrycznych oraz innych urządzeń wymagających zasilania. Opisana metoda opiera się na obserwacji zachowań projektanta w trakcie tworzenia dokumentacji i odwzorowanie ich poprzez skrypt współpracujący z oprogramowaniem Revit. Rezultatem zastosowania proponowanego rozwiązania jest znaczne przyspieszenie prac projektowych, zwiększenie dokładności oraz możliwość wyznaczenia spadków napięć.

Słowa kluczowe: BIM, Revit, automatyzacja, projektowanie, instalacje elektryczne

1. Wstęp

Od kilku lat obserwuje się dynamiczny wzrost zainteresowania metodyką modelowania informacji o budynku BIM. Przy czym, każdy uczestnik procesu budowlanego definiuje to pojęcie w nieco inny sposób [1]:
- Cyfrowy Model obiektu budowlanego (ang. *Building Information Model*),
- Modelowanie informacji o obiekcie budowlanym (ang. *Building Information Modeling*),
- Zarządzanie informacją o obiekcie budowlanym (ang. *Building Information Management*).

Częścią wspólną powyższych definicji jest jedna, spójna baza danych, zawierająca niezbędne informacje geometryczne (wymiary), a także niegeometryczne (dane katalogowe, terminy przeglądów technicznych itp.), określana jako Cyfrowy Model BIM [2]. Model ten jest stale aktualizowany w ciągu całego cyklu życia obiektu, począwszy od projektu koncepcyjnego, poprzez fazy projektów, budowlane jak i zarządzania obiektem, kończąc na jego rozbiórką. Tym samym, BIM pozwala na uporządkowanie i koordynację informacji wymienianych między uczestnikami procesu budowlanego [3]. Istotną zaletą modelu cyfrowego BIM jest możliwość detekcji oraz kontroli ryzyka, w szczególności kolizji instalacyjnych i logistycznych, na wczesnym etapie, co w znacznym stopniu ogranicza koszty budowy.

Obecnie w Ministerstwie Infrastruktury prowadzone są konsultacje społeczne w związku z planami wdrożenia BIM w zamówieniach publicznych. W ich trakcie projektanci zwracają uwagę na brak przygotowania do pracy w tej technologii. Jako powody wskazywany jest brak specjalistów w tym zakresie oraz koszty związane z zakupem nowego oprogramowania. Ważnym aspektem jest również konieczność zwiększenia nakładów pracy potrzebnej do wykonania dokumentacji w BIM, ponieważ oprócz przygotowywanych dotychczas dwuwymiarowych rysunków CAD konieczne będzie zawarcie dodatkowych informacji, co skutkować może zmniejszeniem zysków biur projektowych [4].

Odpowiedzią na zgłaszane wątpliwości jest bez wątpienia automatyzacja prac. Dostępne programy komputerowe, służące do tworzenia modeli 3D, wyposażone są w środowiska programistyczne, dzięki którym możliwe jest tworzenie własnych rozwiązań, optymalizujących projektowanie. W artykule przedstawiono przykład automatyzacji działań projektowych za pomocą autorskiego rozwiązania zaimplementowanego w postaci skryptu Dynamo z wykorzystaniem języka Python i dokumentacji Revit API.

2. Automatyzacja i optymalizacja z wykorzystaniem BIM

Istniejące opracowania bardzo szeroko opisują możliwości wykorzystania BIM w procesie automatyzacji i optymalizacji na wszystkich etapach cyklu życia obiektu. Technologia ta stosowana jest zarówno jako narzędzie wspomagające zarządzanie obiektem budowlanym [5] przez optymalizację kosztową [6], czy optymalizację bilansu energetycznego budynku [6, 7], jak też na etapie prac projektowych. Analizy naprężeń elementów

Autor korespondujący:
Dariusz Rocki, dariusz.rocki@arup.com

Artykuł recenzowany
nadesłany 26.05.2020 r., przyjęty do druku 10.09.2020 r.

konstrukcyjnych przeprowadzane są poprzez integrację Autodesk Robot z MATLAB z Autodesk Robot [8]. BIM wspiera również tworzenie wizualizacji i planów ewakuacyjnych dla obiektów budowlanych za pomocą symulacji [9].

Widoczne jest również zainteresowanie wykorzystaniem w BIM optymalizacyjnych algorytmów genetycznych – m.in. w zakresie optymalizacji zapotrzebowania energetycznego budynków [10], a także sieci neuronowych, które stosowane są m.in. przy detekcji pęknięć w elementach konstrukcyjnych na podstawie wykonanych zdjęć [11].

Narzędzia BIM wykorzystywane są także w zakresie szeroko pojętego zarządzania ryzykiem, obejmującego nie tylko optymalizację wykorzystania przestrzeni placu budowy, ale również bezpieczeństwo osób znajdujących się w obszarze, w którym przeprowadzane są prace budowlane. Zwraca się szczególną uwagę na wykorzystanie rozszerzonej rzeczywistości AR (ang. *augmented reality*) w trakcie trwania samej budowy, jak też na etapie projektowym [10, 11].

Odpowiedzią na zgłaszane przez projektantów obawy związane z wprowadzaniem BIM, jest bez wątpienia automatyzacja prac projektowych. W zakresie instalacji elektrycznych, optymalizacja i automatyzacja przeprowadzana jest zazwyczaj na bazie dokumentacji Revit API [12]. Istniejące opracowania dotyczą głównie prostych operacji związanych z pobieraniem z modelu cyfrowego informacji dotyczących liczby opraw oświetleniowych i ich parametrów, w szczególności sumarycznej mocy [13]. Natomiast wraz z upowszechnianiem się technologii BIM w środowisku budowlanym, powstają narzędzia optymalizacyjne i automatyzujące prace projektowe. Jednym z przykładów takiego rozwiązania jest narzędzie optymalizujące obliczenia natężenia oświetlenia w pomieszczeniach – Elum Tools [14].

Widoczne jest również zainteresowanie wykorzystaniem w BIM optymalizacyjnych metod optymalizacji globalnej, w szczególności algorytmów genetycznych [15]. Wynika to z mocno nieliniowej postaci funkcji celu jak też rozmaitych ograniczeń.

3. Struktura rozwiązania

Celem zadania jest automatyzacja i autorska optymalizacja procesu podłączania odbiorów elektrycznych do obwodów zasilanych ze wskazanej tablicy rozdzielczej. Na rysunku 1 przedstawiono strukturę proponowanego rozwiązania zrealizowaną już praktycznie w postaci oprogramowania zintegrowanego z narzędziami BIM. W programie wykorzystano domyślny węzeł Dynamo [15] – *Element.GetParameterValueByName* oraz węzeł z biblioteki *Rhythm – FamilyInstances.Space.*

Jednocześnie, aby ułatwić praktyczne wykorzystanie proponowanego rozwiązania został stworzony dedykowany interfejs użytkownika, umożliwiający właściwe zastosowanie narzędzia automatyzującego.

3.1. Stosowane węzły Dynamo

Panel Connector korzysta z węzła *Element.GetParameterValueByName*, pozwalającego na uzyskanie informacji o konkretnym parametrze danego elementu. Węzeł ten otrzymując na wejściu zmienna (lub lista zmiennych) typu *element*, oraz nazwę poszukiwanego parametru podaną jako zmienna typu *tekst*, zwraca wartość (lub listę wartości) w postaci zmiennej tekstowej.

Drugim stosowanym w skrypcie węzłem, jest znajdujący się w bibliotece Rhythm węzeł *FamilyInstances.Space.* Daną wejściową jest w tym przypadku zmienna (lub lista zmiennych) typu *element.* Na wyjściu otrzymujemy natomiast zmienną (lub listę zmiennych) typu tekstowego.

Wykorzystane węzły Dynamo pozwalają na wprowadzenie do programu ostatniej danej wejściowej, jaką są nazwy pomieszczeń, w których znajdują się elementy, w postaci listy zmiennych tekstowych.

3.2. Główna część programu

Na główną część rozwiązania składają się dwa węzły Dynamo napisane w języku IronPython 2.7.

3.2.1. Rozłączanie istniejących obwodów

Próba podłączenia do obwodu elektrycznego elementu, który został uprzednio podłączony do innego obwodu skutkuje generowaniem przez Revit nieprecyzyjnych komunikatów o wystąpieniu błędów ogólnych. Zastosowanym rozwiązaniem powyższego problemu jest odłączenie wybranych przez użytkownika elementów od ich dotychczasowych obwodów.

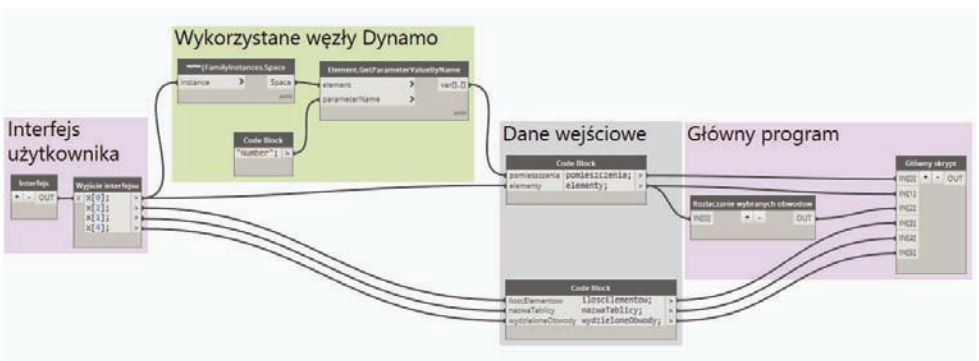
Opracowany węzeł otrzymuje na wejściu listę elementów wybranych przez użytkownika, a następnie za pomocą kolektora pobiera z modelu listę wszystkich istniejących obwodów elektrycznych. Zmienna *wszystkieObwody* jest listą danych typu *element*:

```
wszystkieObwody = FilteredElementCollector(doc).
OfCategory(BuiltInCategory.OST_ElectricalCircuits).
```

W kolejnym kroku identyfikowane są obwody, od których należy odłączyć wybrane elementy. Odłączanie elementów od obwodów umożliwia metoda *RemoveFromCircuit* [12]:

```
for obwod in wszystkieObwody:
   for element in wybraneElementy:
      if obwod.Name == element.LookupParameter('Circuit
                                    Number').AsString():
         obwod.RemoveFromCircuit(element)
```

W celu zapewnienia odpowiedniej kolejności wykonywanych operacji, węzeł na wyjściu zwraca wartość True przekazywaną do głównego skryptu (por. rys. 1).



Rys. 1. Struktura programu
Fig. 1. Program structure

3.2.2. Główna procedura

Przekazana z interfejsu użytkownika zmienna wejściowa *wydzieloneObwody* zawiera listę danych tekstowych, określających elementy, które powinny zostać podłączone do oddzielnych obwodów elektrycznych. Skrypt sprawdza zgodność danych zawartych w zmiennej *wydzieloneObwody* z wartościami zapisanymi w parametrze *Comments* wybranych przez użytkownika elementów

```
for element in elementy:
   if UnwrapElement(element).
      LookupParameter('Comments').AsString()
      in wydzieloneObwody:
```

Elementy, u których zgodność zostanie wykryta zostają przeniesione do listy *zestawWydzielony.*

Pozostałe elementy przekazywane są do opracowanego autorskiego algorytmu sortującego, który przeprowadza dwukrotne sortowanie – w pierwszej kolejności zgodnie z numeracją pomieszczeń, a następnie w zależności od sposobu montażu elementów (montaż ścienny, nastropowy lub w puszce podłogowej).

Następnym krokiem jest uwzględnienie sytuacji, w której w dwu lub większej liczbie kolejnych pomieszczeń, liczba elementów pozwala na podłączenie ich do wspólnego obwodu elektrycznego.

Zakładając, że:

n – liczba pomieszczeń,

g – podana przez użytkownika maksymalna liczba elementów podłączonych do jednego obwodu,

i_1, i_2, ..., i_n – liczba gniazd w poszczególnych pomieszczeniach, przy czym $i_1 \neq i_2 \neq \ldots \neq i_n \neq g$.

Lista elementów przyjmuje postać:

Tabela 1. Struktura listy
Table 1. List structure

pomieszczenie1	[pomieszczenie1, element_ID11] [pomieszczenie1, element_ID12] … [pomieszczenie1, element_ID1i$_1$]
pomieszczenie2	[pomieszczenie2, element_ID21] [pomieszczenie2, element_ID22] … [pomieszczenie2, element_ID2i$_2$]
…	…
pomieszczenieN	[pomieszczenieN, element_IDn1] [pomieszczenieN, element_IDn2] … [pomieszczenieN, element_IDni$_n$]

Zasada działania:

a. Dla każdego pomieszczenia pobierz wartość i – liczba gniazd w pomieszczeniu.

b. Jeżeli $i \geq g$, wówczas i zostaje podzielone przez g. Część całkowita wyniku dzielenia jest liczbą obwodów w pomieszczeniu. Reszta z dzielenia, którą nazwiemy i_{mod}, określi liczbę elementów, które będą podłączone do dodatkowego obwodu. W celu określenia i_{mod}, wykonujemy dzielenie modulo liczby gniazd w pomieszczeniu i przez maksymalną liczbę elementów podłączanych do jednego obwodu g.

$$i_{mod} = i\%g$$

Liczba elementów podłączonych do dodatkowego obwodu elektrycznego jest mniejsza od maksymalnej liczby elementów, wobec czego istnieje możliwość podłączenia do niego elementów znajdujących się w kolejnym pomieszczeniu – w tym przypadku korzystamy z zasady opisanej w kolejnym podpunkcie – dla $i < g$.

c. Jeżeli $i < g$ wszystkie gniazda w pomieszczeniu podłączane są do jednego obwodu elektrycznego. Najczęściej podłączenie wszystkich gniazd w pomieszczeniu nie spowoduje wykorzystania wszystkich miejsc w obwodzie elektrycznym. Wobec tego wykonujemy sprawdzenie, czy liczba gniazd w kolejnym pomieszczeniu jest na tyle mała, że możemy podłączyć je pod ten sam obwód. Zastosowano tu pętlę z prostą funkcją warunkową oraz sprawdzeniem, czy dane pomieszczenie nie jest ostatnim w liście:

```
for i in range(len(ilosc)):
       if i+1 <= len(iloscPomieszczen):
                 if ilosc[i] + ilosc[i+1] < g
```

Wyniki opisanego algorytmu przechowywane są wewnątrz listy:

```
elementyPonumerowane = []
```

w której każdy element jest listą opisaną jako:

```
elementyPonumerowane[n] = [nrObwodu, [IDelementu1,
                                      IDelementu2,
                                      …,
                                      IDelementuN]]
```

Jak wspomniano we wstępie, jedną z różnic między dotychczasowym sposobem projektowania – CAD, a technologią BIM, jest możliwość modelowania i zarządzania informacjami z poziomu jednej, kompletnej bazy danych. Projektowanie za pomocą narzędzi CAD wymagało wielokrotnego, manualnego wprowadzania tych samych informacji na rzutach kondygnacji oraz schematach tablic elektrycznych, powodując bardzo często błędy polegające na braku spójności między rysunkami.

Cyfrowy Model BIM umożliwia wielokrotne wykorzystanie raz wprowadzonej informacji, w szczególności podłączenie elementu widocznego na rzucie kondygnacji do nowego obwodu elektrycznego, powoduje automatyczną aktualizację zestawienia obwodów w tablicy elektrycznej, które program Revit przygotowuje w formie tabelarycznej.

W programie Revit, obwody elektryczne występują jako klasa *ElectricalSystem* należąca do przestrzeni *Autodesk.Revit.DB.Electrical.Namespace* [12]. Opisywany program Panel Connector, wykorzystuje tę klasę oraz dwie jej metody: *Create,* oraz *SelectPanel.*

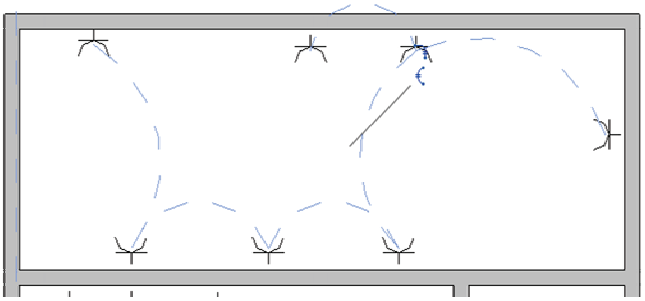
W celu stworzenia nowego obwodu elektrycznego zastosowana została metoda *Create* klasy *ElectricalSystem*, należącej do przestrzeni *Autodesk.Revit.DB.Electrical.Namespace* [12]. Metoda ta użyta została z argumentem *ElectricalSystemType*, określającym typ systemu elektrycznego. Program Revit umożliwia zastosowanie systemów typu:

UndefinedSystemType – system niezdefiniowany,
Data – stosowany do łączenia elementów sieci komputerowych,
PowerCircuit – system obwodów elektrycznych – wykorzystany w opisywanym programie,
Telephone – system telefoniczny,
Security – system bezpieczeństwa,
FireAlarm – system alarmu pożarowego,
NurseCall – system przywoławczy,
Controls – system typu controls,
PowerBalanced – system zbalansowany,
PowerUnBalanced – system niezbalansowany [12].

W opisywanym rozwiązaniu, nowy obwód elektryczny tworzony jest przez:

```
doc = DocumentManager.Instance.CurrentDBDocument
input = UnwrapElement(listaElementow)
for zestaw in input:
       electComponents = List[ElementID]()
       for i in zestaw:
              electCompoments.Add(i.Id)

nowyObwod = ElectricalSystem.Create(doc,
            electComponents,
            ElectricalSystemType.PowerCircuit)
```



Rys. 2. Stworzony system bez podłączenia do tablicy elektrycznej
Fig. 2. Created system not connected to distribution board

Stworzony system stanowi zbiór powiązanych ze sobą elementów, który w programie Revit widoczny jest w postaci elementów połączonych krzywą przerywaną (por rys. 2).

Podłączenie nowego obwodu do wskazanej przez użytkownika tablicy elektrycznej odbywa się za pomocą metody *SelectPanel*:

```
panel = UnwrapElement(wybranaTablica)
newcircuit.SelectPanel(panel)
```

W efekcie powyższych operacji otrzymujemy obwód elektryczny widoczny zarówno na rzutach kondygnacji, jak też w zestawieniu obwodów elektrycznych.

3.3. Transakcje

Modyfikowanie elementów lub samego modelu Revit możliwe jest z poziomu IronPython przez transakcje, czyli ciąg operacji, cechujących się właściwościami ACID [18]:

- *Atomicity* – transakcja wykonywana jest jedynie w całości,
- *Consistency* – po zatwierdzeniu transakcji muszą być spełnione wszystkie warunki poprawności nałożone na bazę danych,
- *Isolation* – efekt równoległego wykonania dwu lub więcej transakcji musi być szeregowalny,
- *Durability* – w bazie danych na stałe pozostają efekty pozytywnie zakończonej transakcji.

W trakcie pisania programu, pod uwagę brane były dwa podstawowe sposoby zastosowania transakcji. Pierwszy polega na objęcie jedną transakcją całego procesu podłączania elementów do obwodów elektrycznych. Zaletą tego rozwiązania jest brak konieczności wielokrotnego zamykania i otwierania transakcji, co powinno skrócić czas wykonywania programu. Dodatkowym argumentem przemawiającym za takim rozwiązaniem jest łatwość cofnięcia całego efektu pracy programu

Drugą, ostatecznie wybraną metodą, jest otwieranie transakcji przed każdym podłączeniem obwodu do tablicy i zamykanie jej tuż po wykonaniu tego podłączenia. Decydujące znaczenie miał w tym przypadku fakt, że dzięki takiemu działaniu, użytkownik dostaje możliwość wycofywania efektów działania programu etapami, a nie tylko w całości. Dodatkowo przeprowadzone testy wykazały zauważalną poprawę stabilności programu w przypadku otwierania i zamykania mniejszych transakcji.

Transakcje obsługiwane w programie są za pomocą *TransactionManager*:

```
clr.AddReference('RevitServices')
import RevitServices
from RevitServices.Persistence import
  DocumentManager
from RevitServices.Transactions import
  TransactionManager
doc = DocumentManager.Instance.
  CurrentDBDocument
```

Otwarcie transakcji:

```
TransactionManager.Instance.
  EnsureInTransaction(doc)
```

Zamknięcie transakcji:

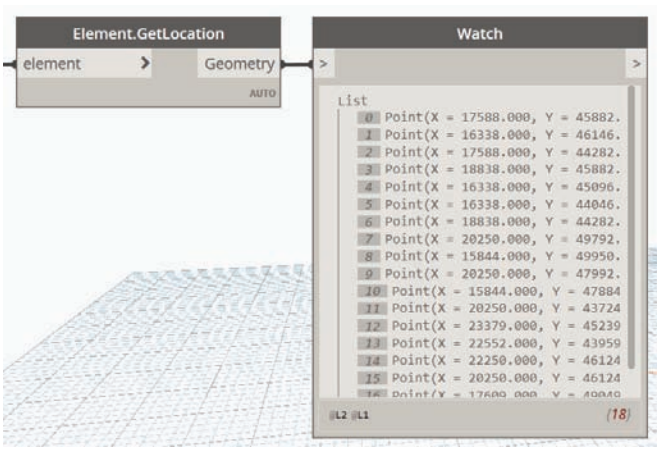
```
TransactionManager.Instance.
  TransactionTaskDone()
```

3.4. Optymalizacja w zakresie długości obwodów

Przeprowadzone testy wykazały poprawność działania programu w zakresie podłączania gniazd elektrycznych i opraw oświetleniowych do obwodów elektrycznych w małych pomieszczeniach. Zaobserwowane zostały jednak trudności w przypadku pomieszczeń biurowych typu open space, w których otrzymane obwody elektryczne rozmieszczane były w sposób chaotyczny. Przyczyną tych nieprawidłowości jest porządkowanie elementów zgodnie z ich numerami identyfikacyjnymi, które Revit nadaje automatycznie według kolejności wstawiania elementów do modelu. Zjawisko to obserwowano w pomieszczeniach, w których liczba elementów przekraczała założoną przez użytkownika maksymalną liczbę elementów podłączanych do jednego obwodu elektrycznego.

Możliwe jest w takich przypadkach zastosowanie manualnego podziału przestrzeni, w której znajdują się elementy, jednak nie jest to działanie optymalne, ponieważ może ono powodować błędne wyświetlanie opisów pomieszczeń na rzutach kondygnacji, a jednocześnie, szczególnie w przypadku dużych przestrzeni biurowych, przeprowadzenie takiego podziału jest czasochłonne.

W celu optymalizacji powyższego problemu, zastosowane zostało porównywanie odległości elementów od tablicy elektrycznej, przy czym odległość ta liczona jest jako suma odległości wzdłuż osi X, Y, Z. Dla każdego elementu, a także dla wskazanej tablicy elektrycznej, za pomocą standardowego węzła Dynamo – *Element.GetLocation* – została pobrana informacja dotycząca jego lokalizacji w modelu.



Rys. 3. Węzeł Element.GetLocation w Dynamo
Fig. 3. Dynamo node Element.GetLocation

Dane wyjściowe węzła Element.GetLocation stanowi lista wartości typu *obiekt geometryczny*. W celu uzyskania współrzędnych x, y, z otrzymanych obiektów geometrycznych, zastosowano węzły Dynamo Do otrzymania współrzędnych X, Y, Z elementów zastosowano węzły *Point.X*, *Point.Y* oraz *Point.Z*, dające na wyjściu listy wartości typu *double*.

Otrzymane wartości wstawione zostały do listy *listaElemOdl*, która składa się z par – ID elementu, oraz odległość elementu od tablicy elektrycznej.

```
from math import fabs
listaElemOdl = []
for i in range(len(elemX)):
       if elemX[i]*tabX >= 0:
              roznX = fabs(elemX[i]-tabX)
       else:
              roznX = fabs(elemX[i]+tabX)
       if elemY[i]*tabY >= 0:
              roznY = fabs(elemY[i]-tabY)
       else:
              roznY = fabs(elemY[i]+tabY)
       if elemZ[i]*tabZ >= 0:
              roznZ = fabs(elemZ[i]-tabZ)
       else:
              roznZ = fabs(elemZ[i]+tabZ)
       listaElemOdl.append([elementy[i],
         roznX+roznY+roznZ])
```

W kolejnym kroku lista elementów została uszeregowana względem odległości:

```
listaPosortowana = sort(listaElemOdl,
  key = lambda x:x[1])
```

3.5. Interfejs użytkownika

Interfejs użytkownika stworzony został w języku IronPython 2.7 w oparciu o przestrzeń nazw *System.Windows.Forms*, której dokumentacja dostępna jest na stronie firmy Microsoft [16].

Interfejs użytkownika pobiera z modelu dane wejściowe typu *element* za pomocą kolektora:

```
doc = Document.Manager.Instance.
  CurrentDBDocument
FilteredElementCollector(doc).
  OfCategory(kategoria),
```

który w przypadku tablic elektrycznych przyjmuje postać:

```
tablice = FilteredElementCollector(doc).
  OfCategory(BuiltInCategory.OST_
  ElectricalEquipment),
```

a w przypadku poziomów:

```
poziomy = FilteredElementCollector(doc).
  OfCategory(BuiltInCategory.OST_Levels).
```

W wyświetlonym oknie dialogowym użytkownik modyfikuje dostępne opcje, a następnie interfejs przekazuje do głównego programu następujące dane:

- wybrane przez użytkownika elementy, które program podłączy do obwodów elektrycznych – lista danych typu element,
- maksymalna liczba elementów podłączanych do jednego obwodu – wartość typu liczba naturalna,
- nazwa tablicy elektrycznej, do której podłączone zostaną obwody – wartość typu tekst,
- elementy, które powinny zostać podłączone do wydzielonych obwodów – lista danych typu element.

Konieczność wskazania maksymalnej liczby elementów wynika z praktyki projektowej opartej na paragrafie 22 uchylonego w kwietniu 1995 r. rozporządzenia [17], które ograniczało do 10 maksymalną liczbę gniazd wtyczkowych 230 V podłączonych do jednego obwodu. Inwestorzy bardzo często decydują się na zmniejszenie tej liczby. Ograniczenia liczby opraw oświetleniowych podłączonych do jednego obwodu elektrycznego wynika z rodzaju zastosowanego źródła światła oraz kart katalogowych zabezpieczeń stosowanych w rozdzielnicach elektrycznych.

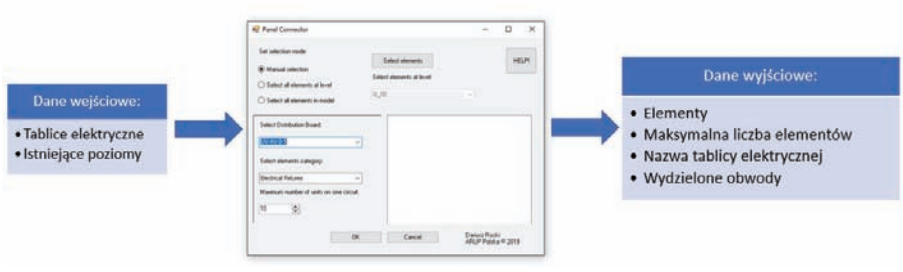
4. Przykład zastosowania

Opisywany program został wykorzystany w trakcie projektu aranżacji przestrzeni biurowej, znajdującej się w wielokondygnacyjnym budynku biurowym na piętrach od trzeciego do szóstego. Powierzchnia użytkowa każdego piętra wynosiła 1372 m^2. W budynku znajdują się dwie klatki schodowe, w których sąsiedztwie zlokalizowane zostały pomieszczenia elektryczne. W każdym z pomieszczeń zaprojektowano po dwie tablice rozdzielcze. Przeznaczeniem tablicy TOS jest zasilenie odbiorów oświetleniowych oraz drobnych odbiorów takich jak porządkowe gniazda wtyczkowe, klimakonwektory, elektryczne podgrzewacze wody itp. Tablica TK zasila natomiast odbiory komputerowe znajdujące się przy biurkach, oraz w serwerowniach piętrowych.

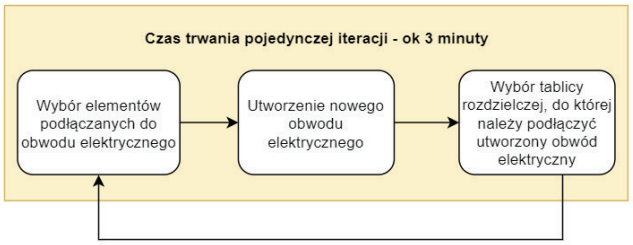
Przeanalizowano dwa sposoby podłączania elementów do obwodów elektrycznych.

Podłączenie elementów do obwodów elektrycznych przy wykorzystaniu wbudowanych narzędzi programu Revit wymaga od użytkownika wyboru elementów, które zostaną podłączone do nowego obwodu elektrycznego, a następnie wyboru tablicy rozdzielczej. Stworzenie pojedynczego obwodu elektrycznego, w zależności od rozmieszczenia i liczby elementów, zajęło 1–7 minut. Dla dalszej analizy przyjęto średni czas, który wyniósł w przybliżeniu 3 minuty.

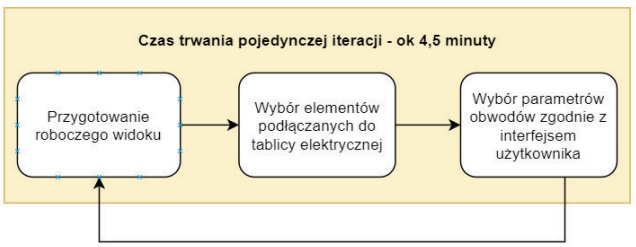
Następnie to samo zadanie wykonano przy wykorzystaniu programu Panel Connector. Elementy podłączane do tablic elektrycznych wybierane były manualnie, a w celu uproszczenia takiego sposobu wybierania elementów, za każdym razem najpierw przygotowywano widok zawierający jedynie elementy podłączane w danej iteracji.



Rys. 4. Struktura interfejsu użytkownika
Fig. 4. User Interface structure



Rys. 5. Podłączanie elementów do obwodów elektrycznych z wykorzystaniem wbudowanych narzędzi Revit
Fig. 5. Connecting elements to electrical circuits using built-in Revit tools



Rys. 6. Podłączanie elementów do obwodów elektrycznych z wykorzystaniem Panel Connector
Fig. 6. Connecting elements to electrical circuits using Panel Connector

Czas potrzebny na podłączenie elementów do tablic elektrycznych zestawiono w poniższej tabeli.

		Czas	
Oznaczenie tablicy elektrycznej	Liczba obwodów	Wbudowane narzędzia Revit	Panel Connector
TOS-3.1	18	54 min	4 min
TK-3.1	23	69 min	5 min
TOS-3.2	21	63 min	4 min
TK-3.2	26	78 min	5 min
TOS-4.1	18	54 min	4 min
TK-4.1	20	60 min	4 min
TOS-4.2	21	63 min	4 min
TK-4.2	29	87 min	6 min
TOS-5.1	18	54 min	5 min
TK-5.1	23	69 min	5 min
TOS-5.2	21	63 min	4 min
TK-5.2	26	78 min	5 min
TOS-6.1	18	54 min	3 min
TK-6.1	20	60 min	4 min
TOS-6.2	21	63 min	5 min
TK-6.2	29	87 min	4 min
weryfikacja działania programu			60 min
	Suma	1056 min	131 min
		17 g 36 min	**2 g 11 min**

Automatyzacja prac projektowych, optymalizująca czas, nie zwalnia projektanta z odpowiedzialności zawodowej, a tym samym, zawsze wymaga weryfikacji. W omawianym przykładzie analiza wyników trwała 60 minut, a czas ten został uwzględniony w tabeli porównawczej. Zaobserwowano jednak znaczną oszczędność czasu potrzebnego na podłączenie elementów do tablic elektrycznych.

Planowane jest dalsze jego rozwijanie programu. W chwili obecnej obliczenia długości przewodów uwzględniają jedynie współrzędne x, y, z zamodelowanych elementów, natomiast pomijany jest rzeczywisty sposób układania przewodów uwzględniający m.in. geometrię ścian. Zwiększenie dokładności umożliwi automatyzację tworzenia zestawień ilościowych przewodów i kabli elektrycznych, a także obliczeń spadków napięć.

5. Podsumowanie i możliwości rozwoju

Opisana zasada działania programu do automatycznego podłączania elementów instalacji elektrycznych zamodelowanych w programie Revit do wybranej przez użytkownika tablicy elektrycznej miała na celu wykazanie, że możliwa jest automatyzacja prac projektowych. Otrzymane wyniki nadal wymagają wiedzy i doświadczenia projektanta, który powinien je ocenić, a następnie wprowadzić korekty tam, gdzie okażą się one niezbędne. Niewątpliwie jednak automatyzacja prac projektowych pozwala ograniczyć czas przeznaczany dotychczas na wykonywanie czynności powtarzalnych, który powinien zostać przeznaczony na kreatywną pracę inżynierską oraz rzetelne sprawdzanie dokumentacji projektowej.

Dotychczasowe obiecujące wyniki wskazują dalsze kierunki badań prowadzących do pełnej optymalizacji projektowania rozprowadzenia sieci elektrycznej w technologii BIM, jak też uczynienie jej odpornej na błędy dokumentacji budowlanej czy też wynikające ze skanowania 3D rzeczywistych pomieszczeń.

Bibliografia

1. Kasznia D., Magiera J., Wierzchowski P., *BIM w praktyce. Standardy, wdrożenie, case study*, Wydawnictwo Naukowe PWN, Warszawa 2017.
2. Rahmani M., Stoupine A., Zarrinmehr S., Yan W., *Optimo: A BIM-based Multi-Objective Optimization Tool Utilizing Visual Programming for High Performance Building Design*, [w:] *Education and Research in Computer Aided Architectural Design in Europe*, Wiedeń, Austria 2015.
3. Andrejczyk E., *Technologia BIM – antidotum na chaos?*, „Materiały budowlane", Nr 7, 2019, 58–60.
4. Tomana A., *Projektowanie w BIM – tańsze czy droższe?*, „Materiały budowlane", Nr 8, 2019, 64–65.
5. Sierra-Aparicio M.V., Ponz-Tienda J.L., Romero-Cortés J.P., *BIM Coordination Oriented to Facility Management*, [w:] *Advances in Informatics and Computing in Civil and Construction Engineering*, 2018.
6. Najjar M., Figueiredo K., Hammad A.W., Haddad A., *Integrated optimization with building information modeling and life cycle assessment for generating energy efficient buildings*, "Applied Energy", Vol. 250, 2019, 1366–1382, DOI: 10.1016/j.apenergy.2019.05.101.
7. Shadram F., Johansson T.D., Lu W., Schade J., Olofsson T., *An integrated BIM-based framework for minimizing embodied energy during building design*, "Energy and Buildings", Vol. 128, 2016, 592–604, DOI: 10.1016/j.enbuild.2016.07.007.
8. Ceré G., Zhao W., Rezgui Y., *Structural Behavior Analysis and Optymalization, Integrating MATLAB with Autodesk Robot*, [w:] *Advances in Informatics and Computing*

in Civil and Construction Engineering, 2018, 379–386, DOI: 10.1007/978-3-030-00220-6_45.
9. Sun Q., Turkan Y., *A BIM Based Simulation Framework for Fire Evacuation Planning*, [w:] *Advances in Informatics and Computing in Civil and Construction Engineering*, 2018, 431–438.
10. Zou Y., Kiviniemi A., Jones S.W., *A review of risk management through BIM and BIM-related technologies*, "Safety Science", Vol. 97, 2017, 88–98, DOI: 10.1016/j.ssci.2015.12.027.
11. Zou Y., Kiviniemi A., Jones S.W., *VUN-based Risk Management: Challenges and Opportunities*, [w:] 23rd CIB W78 Conference, Eindhoven, Netherlands 2015.
12. Revit API, [www.revitapidocs.com].
13. Farooq J., Sharma P., Kumar S., *Applications of Building Information Modeling in Electrical Systems Design*, "Engineering Science and Technology Review", 2017, 119–128, DOI: 10.25103/JESTR.106.16.
14. *ElumTools – Lighting Analysis Add-in for Revit*, "Lighting Analysts. 2020.
15. *Dynamo. Open source graphical programming for design*, [https://dynamobim.org].
16. *System.Windows.Forms*, [https://docs.microsoft.com/pl-pl/dotnet/api/system.windows.forms].
17. Rozporządzenie Ministrów Energetyki i Energii Atomowej oraz Administracji, Gospodarki Terenowej i Ochrony Środowiska z dnia 9 kwietnia 1977 r. w sprawie warunków technicznych, jakim powinny odpowiadać inst. elektroenergetyczne i urządzenia oświetlenia el., Dz. U. 1977 nr 14 poz. 58.
18. Jurkiewicz Z., *Matematyka stosowana. Bazy danych*, Uniwersytet Warszawski, 2013, [http://mst.mimuw.edu.pl/wyklady/bad/wyklad.pdf].
19. Zima K., Mitera-Kiełbasa E., *BIM w zamówieniach publicznych*, „Inżynier Budownictwa", Nr 11, 2018.
20. GitHub, [https://github.com/Amoursol/dynamoPython].
21. Autodesk, *Autodesk Knowledge Network*, [knowledge.autodesk.com].
22. Tang L., Chen C., Tang S., Wu Z., Trofimova P., *Building Information Modeling and Building Performance Optimization*, [w:] "Encyclopedia of Sustainable Technologies", 2017, 311–320, DOI: 10.1016/B978-0-12-409548-9.10200-3.
23. Okakpu A., GhaffarianHoseini A., Tookey J., Haar J., Hoseini A.G., *An optimisation process to motivate effective adoption of BIM for refurbishment of complex buildings in New Zealand*, "Frontiers of Architectural Research", Vol. 8, No. 4, 2019, 646–661, DOI: 10.1016/j.foar.2019.06.008.
24. Jalilzadehazhari E., *Achievien a Trade-Off Construction Solution Using BIM, an Optimization Algorithm, and a Multi-Criteria Decision-Making Method*, "Buildings", 2019, DOI: 10.3390/buildings9040081.

Automation of design using the Panel Connector program

Abstract: This work aims to show ways to facilitate the implementation of BIM in design offices by automation of work. An example program to achieve this goal is a script created in Dynamo environment using Python language and Revit API documentation that allows to automatically connect lighting fixtures, sockets and other devices to a selected switchboard. The described method is based on observing the designer behaviour during the development of documentation and mapping them with a script cooperating with Revit software.

Keywords: BIM, Revit, automation, designing, electrical installations

mgr inż. Dariusz Rocki
dariusz.rocki@arup.com
ORCID: 0000-0002-7460-0688

Absolwent kierunku Elektrotechnika na Wydziale Elektrycznym Politechniki Warszawskiej oraz podyplomowych studiów BIM na Wydziale Geodezji Górniczej i Inżynierii Środowiska Akademii Górniczo-Hutniczej w Krakowie. Aktualnie specjalista ds. BIM i automatyzacji w ARUP Polska oraz doktorant w Zakładzie Automatyki i Inżynierii Oprogramowania Politechniki Warszawskiej. Zainteresowania naukowe obejmują wykorzystanie algorytmów uczących i sieci neuronowych w automatyzacji procesów projektowych.