Representing MEP Systems as Graph data for Rapid Simulation and Analysis

Will Reynolds
Hoare Lea

Learning Objectives

- Discover how MEP systems can be designed as graph data
- Discover how Revit, Forge, and Azure can be used to work with MEP graph data
- Appreciate how the MEP graph data can be stored and queried
- Discover how algorithms can utilize the graph data for rapid simulation and analysis

Description

In this class, we will consider how mechanical, electrical, and plumbing systems can be designed from end to end as graph data. We’ll see how the graph can connect all parts of the system, including spaces and walls, as well as conventional MEP elements. We’ll also go through how the graph can expand to encompass complexity at each stage of the design—from initial conception to design development and, finally, to use. We'll go through an example application of MEP graph data using Revit software, Azure, and the Forge platform. With this application, we'll see how other algorithms can use graph data for rapid simulation and analysis.
## CONTENTS

- Representing MEP Systems as Graph data for Rapid Simulation and Analysis .................. 1
- About the Speaker ........................................................................................................... 3
- Executive Summary ...................................................................................................... 3
- What is Graph data? ...................................................................................................... 4
- How is this useful for MEP? .......................................................................................... 6
  - The geometric context ............................................................................................... 7
  - Versioning ................................................................................................................ 8
  - Design Stages .......................................................................................................... 9
- Applying Graph data practically ..................................................................................... 11
  - Parsing Revit Model to Graph data ......................................................................... 12
- Example uses at particular design stages ..................................................................... 18
  - Use in Concept Design .......................................................................................... 18
  - Use in Construction ................................................................................................. 19
  - Use in Occupation ................................................................................................... 20
- Further Resources ....................................................................................................... 21
About the Speaker

Will has been at Hoare Lea for over 13 years. Originally with a background in electronics and digital systems engineering, Will started at Hoare Lea automating workflows involving AutoCAD and various extranets. Moving swiftly to Revit Will began leading the development of a multitude of enhancements and add-ins, and now has an extensive experience of the Revit API as well design automation knowledge in the context of MEP design.

Executive Summary

Graph data is being used across more and more industries as the preferred store for handing big data. In companion to the presentation this handout explores how graph data technology can be applied to MEP systems; starting from first principals of what graph data is, to why it is useful for MEP systems, and finally how to apply it at the various stages of design.
What is Graph data?

Essentially graph data consists of Node and Edges, where the nodes represent entities and the edges define how they are related. Unlike traditional databases graph data allows for more natural relationships to be defined that closely resemble, or even mirror the real-world object they represent.

Graph data is already being used in a wide variety of industries.
Here’s a simple example of representing the supply and extra systems to a space in graph data. The nodes and relationships illustrate how the graph data is almost identical to the high-level concept of the real system.

A very simple practical (MEP) example:

It is true that existing database technology can also handle this kind of relationships within their own relational schema, however, the key difference is in how the data is queried.

RDBMS Vs. NoSQL Vs. Graph

- Critically, the advantage is in **convenience** and **speed** by which the data can be queried
- The query language, in this case **Cypher**, plays a big role...

Querying complex relationships with SQL can lead to complex queries which are tightly coupled to the relationships defined on the tables, making it difficult to build a system which is maintainable across a varying set of data. Likewise, NoSQL, which although well suited to varying entity schemas, still doesn’t work well for complex relationships.
How is this useful for MEP?

- Discover how MEP systems can be designed as graph data

From the list of published existing industries using graph data that can be found from a google search, or the headline industries boasted by Neo4j and Graph DB products, the AEC industry is notably albescent.

In this section will make the case as to why MEP Systems are well suited to graph data.

**MEP systems wouldn’t be systems without connections!**

- Water flow though pipes (LTHW/BCW etc.) = connections
- Drainage systems = connections
- Air flow through ducts (supply/extract) = connections
- Electrical flow in circuits (power/data etc.) = connections
- Energy Flow = connections
- Occupancy movement through spaces = acquaintances

Connections are a principal concern with MEP systems which makes them an excellent fit for graph databases.

**Sneak preview: From a single space to... everything else**
In addition to pretty graphs, there are key advantages to storing MEP systems as graph data.

### Advantages of graph data in MEP

- High level abstractions and the how the data is stored can be the same
- How you think about the design and the way in which it’s modeled are **synonymous**
- Makes it easy to convey your design to others
- Directly convey your design to **machines**
- The graph data scales in unison with the project lifecycle

### The geometric context

Considering how the 3D geometry relates to the entities in the graph data is a critical requirement. It’s not a trivial to implement given the range of possible 3D geometry, from flat planes to fine detailing. It is therefore necessary to derive an algorithm to extract the most significant features whilst ignoring other aspects of the geometry which can be negated.

### What about the geometric context?

Geometric constraints are also very important, how can the graph data capture this?

- Simplify geometry into nodes which represent surfaces, areas and volumes
  - Extract most significant features from the geometry
  - Ignore insignificant features
- Location information can be included in the graph
Versioning

Additional versions of the same model which are published to the same graph database can keep relationships to any unchanged elements. This makes it possible to query the data and compare parameter values across different models, and track in which models the elements exits.

**What about the versioning?**

How can the graph data capture different versions as the design evolves?

- Edges are cheap! Relationships to previous versions of elements can be linked together
- File locations/URIs/identities, revisions, date and time information can all be stored.
Design Stages

As the design develops the graph data can scale to encompass the detail at each stage. Relationships can be added to associate the detailed design with the original concept elements.

In this way it is possible to build concept design before it can exist as elements in a 3D model. This can capture standard design concepts which can be simulated and developed in isolation, and incorporate where required into the concept design, and ultimately detailed design, of project model.

Graph Data in Concept design

- Create **systems** and connect them up **before** they exist in Revit model
- **Calculation** methods of your **choice**
- **Modularise** your systems; reusable graphs of standard, designs, building types, and plant equipment
Where the graph encompasses design information alongside geometry information it allows for mathematical models to be derived from the graph. A purely analytical representation of all systems involved is readily available to simulation algorithms. Input parameters and coefficients can be adjusted real-time, as well as adjustment to the graph structure itself.

This is especially applicable where the graph includes feature extracted geometry data. In this case it can make the geometry available in an analytical form which can be used directly instead of translating raw geometry between applications.

**Using the graph data for Analysis and Simulation**

- **Key part:** being able to **easily query the data**
- **Complex geometry becomes simpler**
- Simulation algorithms don’t need to keep parsing geometry
- Can be read and written to by other software (e.g. IES, TAS, Sefaira, Open Studio)
- **Reuse** graph data from previous projects.

Graph data is widely used in the context of machine learning, and MEP graph data can be too. Once such application could be: given the bonding features (area, facing vector, length and height, etc.) learn the performance (e.g. heating/cooling load) for a different set of bounding feature values.

**Graph data and Machine Learning**

- Read-made features are present in the graph, e.g. space surface **boundings**
- Together with complex simulations (e.g. from CFD simulations) from the raw geometry, or, more pertinently, from in use data.
- ML can be used to learn how the graph would perform, for a different set of feature values, without running computationally intensive tasks again.
Applying Graph data practically

- Discover how Revit, Forge, and Azure can be used to work with MEP graph data
- Appreciate how the MEP graph data can be stored and queried
- Discover how algorithms can utilize the graph data for rapid simulation and analysis

This section will explore how the concept of applying Graph data to MEP systems can be applied practically with existing technology.

Existing Graph Database Technology

This is a fast developing area.
I’ve chosen Neo4j (https://neo4j.com)
- Large community
- Well documented
- Active development

Native: Graphs data relations sorted directly in storage optimized for graph data. Index free adjacency.
Non-Native: Data is serialized to other relational storage type.

Practical Demo: Architecture
Parsing Revit Model to Graph data

With the basic architecture in mind here are 5 steps to illustrate how a full application could upload the graph data from Revit, processes it, and write data back to Revit.

**Practical Implementation**

Design development with an existing Revit model.

- **STEP 1**: Setting up neo4j and .Net or JavaScript drivers
- **STEP 2**: Converting a Revit model to graph data
- **STEP 3**: Visualizing the data
- **STEP 4**: Analyzing the data
- **STEP 5**: Writing Graph data to back to Revit

The first step in getting started is to get Neo4j up and running. Thankfully, since we’re using docker, it’s just a matter of a few commands:

**STEP 1 : Setting up neo4j and .Net or JavaScript drivers**

- Create neo4j docker image
  - Install Docker
  - `$ docker pull neo4j:latest`
- Start neo4j image locally
  - Open browser at `https://localhost:7473`
- Obtain .Net and JavaScript neo4j drivers
  - `$ nuget install Neo4jClient`
  - `$ npm install --save neode`
- Optional: Push neo4j docker image to azure container (and start instance)
Now on to pushing some data from Revit to the Neo4j database. Admittedly, some programming is required from this point on, but it could be all achievable in Dynamo with some custom nodes or Python script. The data in the Neo4j database can be used directly through its web portal and/or pulled into Power BI or Excel.

Here are some considerations to use when parsing Revit model data to graph data.

### Step 2: Parsing a Revit model to Graph data

**MEP Elements and connections**

- For Circuits, Ducts, Pipes, Cable Tray and their fittings and accessories

  - Get all the elements from the model
  - For each element:
    - Add element to graph and find all its connected elements
    - Add connections such as the spaces and levels each element is in or on
    - Follow onto connected elements and add connection to graph
  - Ignore elements which have already been visited

- Traversal algorithm
  - Iterative rather than recursive
  - Two stage: in-memory graph first, then write to API or graph DB directly

- *Could* be all performed in Dynamo with custom nodes (none exist yet), or a bit of Python/C#
Electrical systems can be connected together with “ELECTRICAL_FLOW_TO” relationship, starting from the first DB panel all the way to the final outlet or light fitting.

**Step 2 : Parsing a Revit model to Graph data**

**Electrical Systems**

MATCH p=(n:DBPanel)-[r:ELECTRICAL_FLOW_TO*]->(s:Circuit)-[z:ELECTRICAL_FLOW_TO]->(b) RETURN p LIMIT 300

Mechanical systems can be connected with “AIR_FLOW_TO”, or “HYDRONIC_FLOW_TO” relationships between fittings, ducts, pipes, accessories and equipment.

**Step 2 : Parsing a Revit model to Graph data**

**Mechanical Systems**

MATCH p=(n:Space {Number:"01-01"})-[:FLOWS_TO_SPACE]->(m:Terminal)-[:AIR_FLOW_TO*1..20]->(s:Terminal)-[:FLOWS_TO_SPACE]->(n:Space {Number:"01-02"}) RETURN p LIMIT 30
Space Geometry is a little more difficult is we want to use a feature extraction algorithm. Essentially each surface between any two spaces can be connected via a “Surface” node and “BOUNDED_BY” relationships. The Surface has an “IS_ON” relationship to the wall on which the surface resides, and then in turn the wall has an “IS_OF” relationship to the wall type.

**Step 2 : Parsing a Revit model to Graph data**

*Spaces and Surfaces*

![Graph model](image)

MATCH (n:Space {Number:"01-08")-[:BOUNDED_BY]->(s:Section)-[:BOUNDED_BY]-(p:Space {Number:"01-10")})
MATCH (s)-[:IS_ON]-(m)-[:IS_OF]-(t)
RETURN n,p,s,m,t LIMIT 30

**Step 2 : Parsing a Revit model to Graph data**

*Geometric feature extraction*

- Brute force ray trace method
  - For each space bounding surface
    - Divide face into i U V point grid
    - Each point area weight = face surface area / number of points
    - Get normal vector and origin at UV point on face
    - Construct ray and get intersect with other space faces
    - Get intermediate element (wall/floor/window)
    - Ignore rays which hit the same space
  - Group ray hits by space, intermediate element and intersecting face
    - Bounding area is total of ray weights
    - Put nodes in to the graph and connect them up
  - Performance optimisations
    - Put geometry into Octree for more efficient ray check
    - Favour the last hit face when intersecting the next ray
    - Group rays into buckets of similar orientation (segmented sphere)
Using the Forge Viewer to display data alongside the graph data is where the Node.js app comes into play. Here we have utilised the ExternalId present in the Forge model with the UniqueId stored in the graph data. This allows selection and highlighting between data from the graph database and the model elements present in the Forge viewer.

**Step 3 : Visualizing the data**

Displaying data alongside model in the Forge Viewer

- Get all spaces for model from the graph DB
- Match elements by UniqueId to ExternalId
- Isolated matched elements in Forge viewer

Using Power BI we can draw data from the graph database for custom charts, graphs, and other data visualizations and analysis. A simple Power Query can be used to connect the database and generate dynamic tabulated data.

**Step 4 : Analyzing the data**

Breaking out to Power BI

- Space element count verification
- Verify elements are fully connected
- Visualize heating/cooling loads
Writing back data to the Revit model is just a matter of matching the UniqueIDs against the original elements, and setting any changed parameters. Change control could be done within the graph by means of a “ChangeRequest” node which would contain details of the changed parameters. The changes requests could then be reviewed against the Revit model and accepted or rejected by the Revit user, or other design team members.

- Once there is a separate database the update + write back workflow is possible
- Similar to Jeremy Tammiks’s example: https://github.com/jeremytammik/FireRatingCloud/
- Relationship of graph nodes to model elements is made possible by their IDs
- Accept or Reject change management can be implemented in graph data
Example uses at particular design stages

Use in Concept Design

Here is an example of using a basic configuration interface for an LTHW plant system. The graph database contains the elements specified through the configurator, but at this stage these elements are not yet present in any Revit model. This allows a schematic to be built in advance, and then later associated with Revit model elements.
Use in Construction

An example use of the graph data at construction stage is to associate the original design elements to the fabrication parts. This allows the original design information to be retained and carried forward to the fabrication parts which don't have inherent system properties and calculations.

**Example use in Construction**

- Associate fabrication parts with design elements
- Maintain systems relationships
- Validate fabrication against design intent
Use in Occupation

At the occupation stage, where a building may have sensors reporting temperatures, occupancy, and other information, the sensors can be added to database along with connections to the spaces and model elements to which they relate. With these relationships it allows harvesting and displaying of sensor data in context of the 3D model, together with other design parameters about the mechanical and electrical systems.

Example use in Occupation

- Visualise systems
- Validate in use data against design data
- Feed back performance into future designs

Machine learning
Further Resources

- Free eBooks on Graph Databases and Neo4j graph algorithms:
  

- Online of courses:

- My GitHub repository:
  [https://github.com/willhl/GraphData-MEP](https://github.com/willhl/GraphData-MEP)

- Neo4j Client Packages:
  - NuGet:
    - [https://github.com/neo4j/neo4j-dotnet-driver](https://github.com/neo4j/neo4j-dotnet-driver)
    - [https://github.com/Readify/Neo4jClient](https://github.com/Readify/Neo4jClient)
  - NPM:
    - [https://www.npmjs.com/package/neo4j](https://www.npmjs.com/package/neo4j)
    - [https://www.npmjs.com/package/neode](https://www.npmjs.com/package/neode)