Representing MEP systems as graph data for rapid simulation and analysis

Will Reynolds
Principal Digital Applications Developer
About the speaker

Will Reynolds

- 13 Years at Hoare Lea
- Principal applications developer - Digital Innovation group
- Background in electronics and digital systems
- Document Controller - AutoCAD and extranet processes.
- Software development full time with Revit.

WillReynolds@HoareLea.com
https://twitter.com/d2liYmxl
https://www.instagram.com/d2liymxl/
Agenda

What is Graph Data?
Why Graph Data in MEP?
How to apply it?

Construction Stage
Design Stage
Concept Stage
Occupation Stage
Construction Stage

What is Graph Data?
It’s not about Graphs or Charts
It’s all about connections

- It has Nodes, connected by Edges
- Edges can be directional
- Nodes can have multiple labels and multiple properties
- Edges have a type and multiple properties (weights)

It’s a database, but,
- Does not require a schema
- Highly optimized for graph traversal
- Incredibly efficient at querying complex related data
Graph data in existing industries

- Facebook
- Twitter
- Google
- Microsoft
- Ebay

Fraud Detection
Financial
IT Operations
AI and Machine Learning
Social
A very simple practical (MEP) example:
How is this useful for MEP?
 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 = connections

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
Advantages of graph data in MEP

- How you think about the design and the way in which it’s modeled are synonymous
- **Focus** on the **Design**, not the software!
- 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**
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
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
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.
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.
OK I get it, but how can it be practically applied?
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 and Analyzing the data

• **STEP 4**: Writing Graph data to back to Revit
STEP 1 : Setting up neo4j and .Net or JavaScript drivers

• Create neo4j docker image
  o Install Docker
  o $ docker pull neo4j:latest
• Start neo4j image locally
  o Open browser at https://localhost:7473
• Obtain .Net and JavaScript neo4j drivers
  o > nuget install Neo4jClient
  o $ npm install --save neode
• Optional: Push neo4j docker image to azure container (and start instance)
Step 2: Parsing a Revit model to Graph data

MEP Elements and connections

- For Circuits, Ducts, Pipes, Cable Tray and their fittings and accessories
  - For each element in the model:
    - 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#
Step 2: Parsing a Revit model to Graph data

Electrical Systems

All connections from fixtures to DB Panels

MATCH p=(n:DBPanel)-[r:ELECTRICAL_FLOW_TO*]->(s:Circuit)-[z:ELECTRICAL_FLOW_TO]->(b) RETURN p LIMIT 300
Step 2 : Parsing a Revit model to Graph data

Mechanical Systems

Duct connections room to room

MATCH p=(n:Space {Number:"01-01"})<-[:FLOWS_TO_SPACE]-(:Terminal)-[:AIR_FLOW_TO*1..20]-(:Terminal)-[:FLOWS_TO_SPACE]->(:Space {Number:"01-02"})
RETURN p LIMIT 30
Step 2: Parsing a Revit model to Graph data
Spaces and Surfaces

Surfaces between rooms

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 UV 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/door/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)
Step 3 : Visualizing and Analyzing 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
Step 3 : Visualizing and Analyzing the data

Breaking out to Power BI

- Cypher queries can return tables
- Cypher supports aggregation and math functions
- Custom Query for Power BI and Excel
- Space element count verification
- Verify elements are fully connected
- Visualize heating/cooling loads
Step 4: Writing Graph data to back to Revit

• Once there is a separate database, the update + write back workflow is possible.

• Similar to Jeremy Tammik'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 use in Concept design

Schematics
Example use in Construction

• Associate fabrication parts with design elements
• Maintain systems relationships
• Validate fabrication against design intent
Example use in Occupation

- Connect sensors to elements in the “digital twin” model
- Validate in use data against design data
- Feed back performance into future designs
Agenda: Revisited

Simple yet powerful

Perfect fit for MEP

Enormous potential

- Design Stage
- Concept Stage
- Construction Stage
- Occupation Stage

?
Resources

• Free ebooks on Graph Databases and Neo4j graph algorithms:
  https://neo4j.com/graph-databases-book
  https://neo4j.com/whitepapers/graph-algorithms-neo4j-ebook/

• Online of courses:
  https://www.linkedin.com/learning/search?keywords=neo4j
  https://www.coursera.org/learn/big-data-graph-analytics

• My GitHub repository:
  https://github.com/willhl/GraphData-MEP

• Neo4j Client Packages:
  NPM: https://www.npmjs.com/package/neode
Questions?