Graph Representation for Obsolescence Issues and Vulnerabilities

Michael S. Bandor

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM21-0623

Agenda

Problem Space

Property Graph Representation

Modeling the Problem

Possible Scenarios

An Approach Using Neo4j

Graph Representation for Obsolescence Issues & Vulnerabilities

Problem Space

Problem Space

Problem: Tracking COTS, GOTS and FOSS obsolescence issues and vulnerabilities in software intensive systems

Questions that need to be asked:

- *What is affected?*
- *When is the product no longer supported?*
- *What is the immediate impact?*
- *What are the secondary and tertiary dependencies?*
- *Where are the products located/used in the system and environments?*
- *What are the impacts of identified vulnerabilities (CVEs)?*
- *How do we track this?*

Problem Space

Organizations tend to use whatever is available (typically a spreadsheet) to identify and track the issues

Tracking relationships and impacts gets to be a very complicated situation at best

Visualization of the impacts is extremely difficult if even possible

A Design Structure Matrix (DSM)[1] approach would show clusters but not secondary and tertiary dependencies.

	A	B	C	D	E	F	G
Element A	A	1				1	
Element B		B		1			
Element C	1		C				1
Element D				D	1		
Element E		1			E	1	
Element F			1			F	
Element G	1				1		G

[1] This is sometimes also referred to as dependency structure matrix, dependency structure method, dependency source matrix, etc. " https://dsmweb.org/ , https://en.wikipedia.org/wiki/Design_structure_matrix

Graph Representation for Obsolescence Issues & Vulnerabilities

Property Graph Representation

Property Graph Representation

Why a graph representation?

- Everything is naturally connected, networks of people, transactions, supply chains
- *"Graphs form the foundation of modern data and analytics techniques, with capabilities to enhance and improve user collaboration, Machine Learning models, and explainable Artificial Intelligence."– Gartner, "Top 10 Tech Trends in Data and Analytics", 16 Feb 2021*

A property graph lets the problem be represented through Nodes and Relationships of the nodes to each other

Property Graph Representation

Nodes* are the entities in the graph.

- They represent objects (nouns)
- They can hold any number of attributes (key-value pairs) called properties.
- Nodes can be tagged with labels, representing their different roles in your domain.
- Node labels may also serve to attach metadata (such as index or constraint information) to certain nodes.

* *The Property Graph Model*, https://neo4j.com/developer/graph-database/

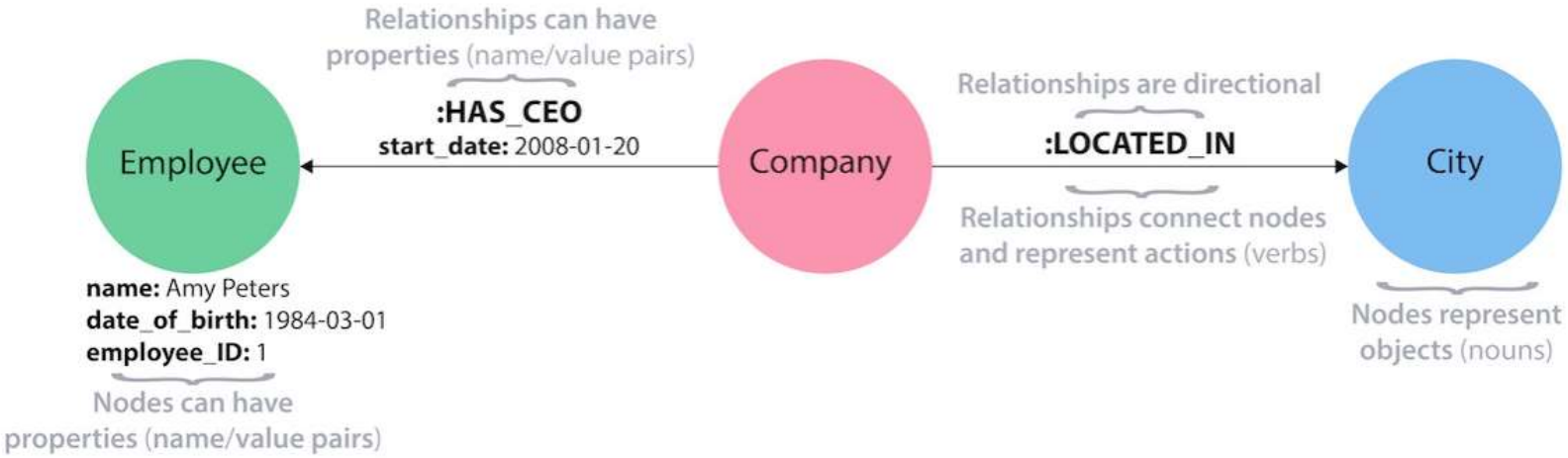
Property Graph Representation

Relationships* provide directed, named, semantically-relevant connections between two node entities (e.g., Employee WORKS_FOR Company).

- Relationships connect nodes and represent actions (verbs)
- A relationship always has a direction, a type, a start node, and an end node.
- Like nodes, relationships can also have properties. In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths.
- Due to the efficient way relationships are stored, two nodes can share any number or type of relationships without sacrificing performance.
- Although they are stored in a specific direction, relationships can always be navigated efficiently in either direction.

* *The Property Graph Model*, https://neo4j.com/developer/graph-database/

Property Graph Representation



* *The Property Graph Model*, https://neo4j.com/developer/graph-database/

Graph Representation for Obsolescence Issues & Vulnerabilities

Modeling the Problem

Modeling the Problem

Using a product centric approach, what information needs to be tracked?

A product has:

- Name & Version
- Manufacturer
- End of Support Date (EOS)
- End of Extended Support Date (EEOS)
- End of Life Date (EOL)
- Category (COTS/GOTS/FOSS)
- Possible dependency on another software product
- Runs on an operating system
- May have one or more vulnerabilities

Modeling the Problem

An operating system has:

- Name & Version
- Manufacturer
- End of Support Date (EOS)
- End of Extended Support Date (EEOS)
- End of Life Date (EOL)

Operating Systems tend to have a different impact on obsolescence issues and vulnerabilities than other software products

- Recommend tracking as separate entity (node) from other software products

Modeling the Problem

Defining the nodes (Product, OS, etc.) and the relationships:

A ***Product***:

- Runs on an OS (RUNS_ON relationship)
- May depend on another product (DEPENDS_ON relationship where it exists)
- Is created by a manufacturer (represented as a node; CREATED_BY relationship)
- Has an obsolescence issue (represented as a node; HAS relationship where an obsolescence issue has been identified and is being tracked externally, i.e., Jira™ ticket)
- Contains one or more vulnerabilities (represented as nodes; CONTAINS relationship where it exists)
- Is installed in one or more environments (represented as nodes; INSTALLED_IN relationship)

Jira is a trademark of Atlassian

Modeling the Problem

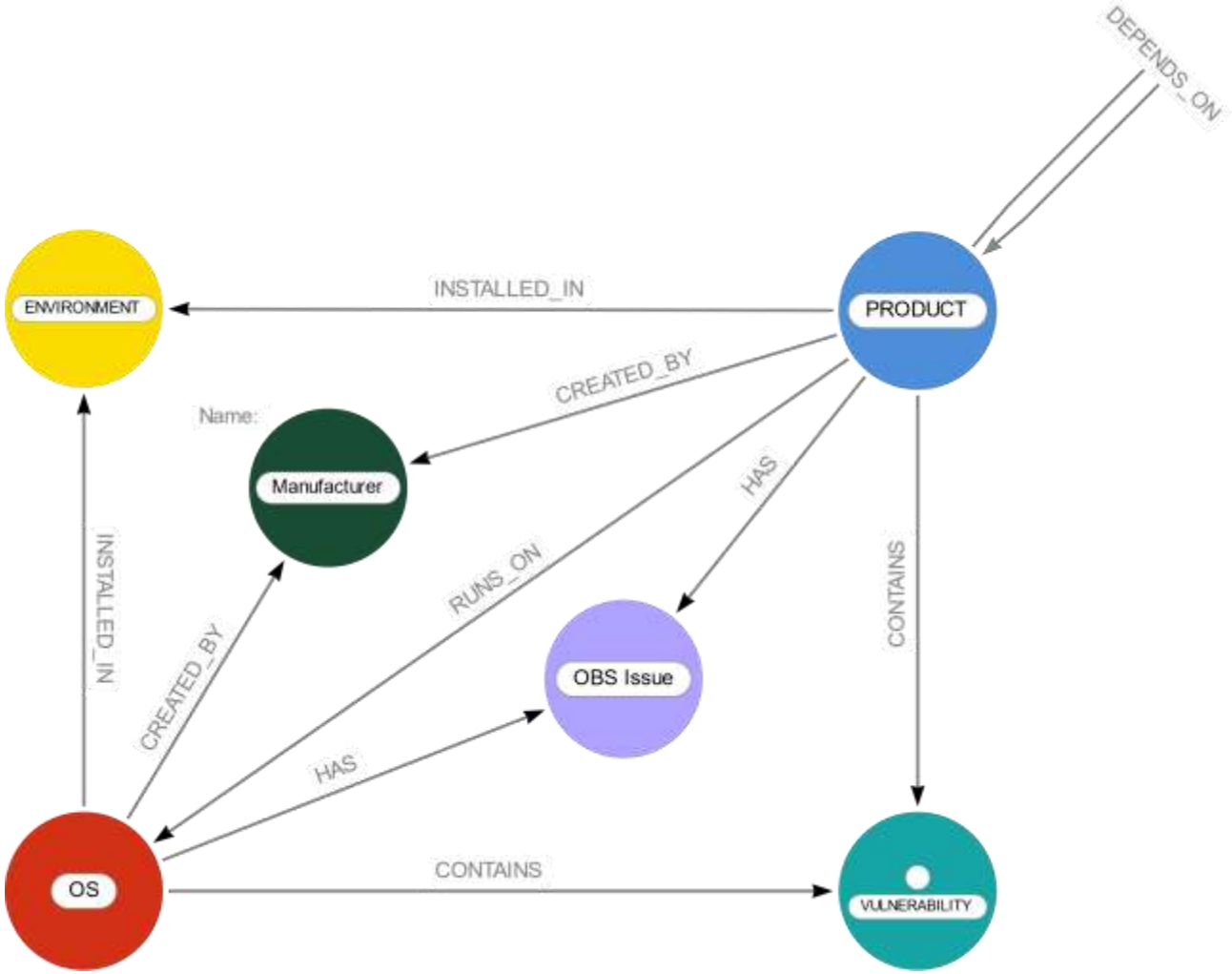
An ***OS***:

- Is created by a manufacturer (represented as a node; CREATED_BY relationship)
- Has an obsolescence issue (represented as a node; HAS relationship where an obsolescence issue has been identified and is being tracked externally, i.e., Jira^{tm} ticket)
- Contains one or more vulnerabilities (represented as nodes; CONTAINS relationship where it exists)
- Is installed in one or more environments (represented as nodes; INSTALLED_IN relationship)

Obsolescence issue is intentionally modeled as a separate node

- Contains a link to a Jira^{tm} ticket
- There may be more than one product covered under a ticket
- Also contains risk information

Modeling the Problem



Graph Representation for Obsolescence Issues & Vulnerabilities

Possible Scenarios

Possible Scenarios

1. A manufacturer announces their software product will no longer be supported next year. What are the impacts to the system?
2. A vulnerability has been identified in a software product. Where is it installed in the system in order to determine the risk?
3. A manufacturer sells a product line to a foreign company. Where is this product in the system and what is the impact?
4. A new environment is going to be established to support product testing. What products need to be on that environment?
5. A newer version of a product is now available. Is there anything keeping the upgrade from happening (dependencies)?

Graph Representation for Obsolescence Issues & Vulnerabilities

An Approach Using Neo4j

An Approach Using Neo4j

What is Neo4j?

Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant[1] transactional backend for your applications. Initial development began in 2003, but it has been publicly available since 2007.

Neo4j is referred to as a native graph database because it efficiently implements the property graph model down to the storage level. This means that the data is stored exactly as you whiteboard it, and the database uses pointers to navigate and traverse the graph.

What is Neo4j https://neo4j.com/developer/graph-database/

[1] *In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps. In the context of databases, a sequence of database operations that satisfies the ACID properties (which can be perceived as a single logical operation on the data) is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction*. https://en.wikipedia.org/wiki/ACID.

An Approach Using Neo4j

Who is using Neo4j?

- MITRE: Cybersecurity Situational awareness (https://neo4j.com/case-studies/mitre/)
- NASA: Lessons learned and knowledge management (https://neo4j.com/users/nasa/)
- Lyft: Data discovery (https://neo4j.com/case-studies/lyft/)
- Lockheed Martin Space: Lifecycle data and parts management (https://neo4j.com/case-studies/lockheed-martin-space/)
- CAST Software: IT architecture visibility (https://neo4j.com/case-studies/cast-software/)
- US Army: Equipment maintenance tracking (https://neo4j.com/case-studies/us-army/)

An Approach Using Neo4j

Working Demo

- Based on existing information captured elsewhere
- Built in about 4 hours including errors and correction of errors without prior knowledge of the tool
- Expanded the input method to natively import the Excel spreadsheets using the built-in APOC[1] library

[1] *Drawing from the unlucky technician in The Matrix movie and the historic Neo4j acronym "A Package Of Components," the name APOC was an obvious choice, which also stands for "Awesome Procedures On Cypher".* https://neo4j.com/blog/intro-user-defined-procedures-apoc/

Questions?

Contact Information

Michael S. Bandor
Senior Software Engineer
Software Engineering Institute (SEI)
Carnegie Mellon University (CMU)
mbandor@sei.cmu.edu
210-380-5563 (work/cell)