Understanding Datasets
Seeing the Unseen through Graph Automations
September 2022
The Need

- Simplified mechanism for building and unifying disparate data sources into a normalize high performance Unified Data Fabric, which can be evaluated and understood for Analytics, Machine Learning, and Simulations
- Mechanisms to visually explore and understand both data entities and their relationships to one another
- Mechanisms to understand and traverse threads of augmented entities
- Mechanisms to project and understand through graph analytics the desired dataset

The Rules Simulation Engine is the Answer, One mechanism for all these things, and more.
Why Graphs?

- High performance Entity<->Entity Relationship Management
- Any Entity can reference any other Entity. Graphs are highly Adaptable, and White Board Friendly (Schemaless)
- Graphs are the foundational element for Analytics, NLP, ML, and AI.
- Graphs are thousands of time faster than the best Relational Databases even on small datasets and the performance gap grows exponentially as dataset sizes increase
- Relationships are First Class Citizens
- Graphs are the cornerstone of our solution

Find Reports and How Many are Managed Three Level Down

**Recursive Cypher**

```
match (boss)-[:Manages*0..3]->(mgr)
where boss.name="Dave Green"
and (mgr-[Manages]->())
```

**Recursive SQL**

```
-- Recursive SQL example
```

Declarative

Describe what you want
Not of how to do it

Graphs are schemaless and adaptable to changing requirements.

Relationships are first class citizens

Comparing Graph and Relational Structures

Dataset of 1,000 people, each with 50 friends. Find all the paths from one person to another in four or less hops.

<table>
<thead>
<tr>
<th># of People</th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDBMS</td>
<td>2000ms</td>
</tr>
<tr>
<td>Neo4J</td>
<td>2ms</td>
</tr>
</tbody>
</table>

Dataset of 1,000,000 people each with 50 friends. Find friends of friends up to five levels deep.

<table>
<thead>
<tr>
<th>Depth</th>
<th>RDBMS (ms)</th>
<th>Neo4J (ms)</th>
<th>Records Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.016</td>
<td>0.010</td>
<td>~2,500</td>
</tr>
<tr>
<td>3</td>
<td>30.267</td>
<td>0.168</td>
<td>~110,000</td>
</tr>
<tr>
<td>4</td>
<td>1543.505</td>
<td>1.359</td>
<td>~600,000</td>
</tr>
<tr>
<td>5</td>
<td>unfinished</td>
<td>2.132</td>
<td>~800,000</td>
</tr>
</tbody>
</table>
Grammar Overview

Grammars are compiled into native graph languages, currently Cypher a Neo4J open standard, but could also generate other Graph languages. Grammars are able to load data from an ever-growing set of data sources.

Integrating Disparate Data Sources

- **Graph Native Language**
- **Compiler**
- **RSE Grammar**
- **Data Source**

RSE Grammars can be compiled into multiple graph languages.

Grammars simplify and augment the ability to create graph structures without having to write and support graph language scripts.

RSE Grammars can import data from a variety of data sources. Currently we support Relational and CSV formats.

Unified Data Fabric Creation Mechanism

Multiple subgraphs defined through Grammars are automatically combined into a single source of truth, a Unified Data Fabric.
Overview Schematic

Rules Simulation Engine Schematic

Easily “See the Unseen” through an easily constructed, high performance, Unified Data Fabric composed from multiple subgraphs. Subgraphs are constructed, from disparate data sources, through our Grammar technologies. Insights from the Unified Data Fabric are enhanced through high powered analytics, machine learning technologies, as well as our Rules Simulation Engine powered by autogenerated domain agnostic Templates.

Overview

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
Making Decisions Through Thread Pathways

Create & Accumulate Graphs

Variable Path, Variable Steps

Simulation Path 1

A -> B -> C -> D -> E

Simulation Path 2

C -> B -> S -> G -> A

Simulation Path 1437

B -> S -> L -> K

Most projects can identify Data Entities that might contribute to the eventual outcome, with new Data Entities constantly emerging.

Our Solution provides a simple mechanism to define scenarios. These scenarios select, and order, Data Entities which can be included into variable length Simulation Paths.

\{A\} -> \{B\} -> \{C\} -> \{D\} -> \{E\},
\{C\} -> \{B\} -> \{S\} -> \{G\} -> \{A\},
\{B\} -> \{S\} -> \{L\} -> \{K\}, ....

Data Entity values may have multiple settings. For example, Income Levels may be described as a set of ranges.

Because of the dynamic nature of projects, Simulation Paths must be easily, and dynamically, defined from external sources.
Data Science Library: A Global View

- **Community Detection**
  - Determines the importance of distinct nodes in the network
  - What are the important predictors that we should consider?

- **Centrality (importance)**
  - Estimates the likelihood of nodes forming relationships
  - Clients with X probably also would choose Y.

- **Similarity**
  - Heuristics Link Prediction

- **Path Finding**
  - Learns graph topology to reduce dimensionality for machine learning
  - Graphs are the base technologies for neural networks, so let us use them to feed our machine learning services.

- **Node Embedding**

- **Group Clustering or Partition Options**
  - Detects how our clients have in common
  - How can we discover emerging group types?

- **Evaluates how alike nodes are**
  - Maybe we don’t need both nodes, simplify our models?
  - Maybe discover new trends?

- **Finds optimal paths. Evaluates route availability and quality**
  - What are the most common pathways through a system, those are the ones we should optimize.
Solution Entity Pallet Mechanisms (CI/CD)

Solution Entities are the cornerstone of our solution.

Solution Entities undergo a Certification Process and normal CI/CD pipeline.

Solution Entities are combined provide perspectives into our high-performance datasets.
The AI Infrastructure Pipeline

Solution Entity Adapters are the core mechanism for integrating disparate components into a collective AI Infrastructure through Declarative Interfaces.

Solution Entities are the next generation, domain agnostic Rule Engine Templates, residing in, and are selected from, a pallet library. Solution Entities expose and are connected through declarative interfaces.
Interface Mechanisms

Current Interface Schema

- Servers and Clients are Roles. A component often operates in both Roles.
- The need for an attribute set, generally requires acquisition through multiple interfaces and multiple requests from the Client.
- The Client then has to parse the presented structures to obtain the desired results.
- The Client is tightly bound to the structure of the Server. Changes to the Server interfaces $(X \rightarrow X_1)$ directly affects all $X$ dependent Clients.

- Tightly bound components are contrary to Plug and Play topologies.
- Interface versioning complicates the topology and the exposing Services.
- Transferring unrequested attributes harms performance on multiple levels.

Declarative Interface Schema

- Each Service interface exposes one or more methods (functions with parameters and return types).
- SOAP, REST, and Language Native, although different, are common interface types.
- Declarative Interfaces establish a common declarative request mechanism throughout the entire topology.

- The Service Exposes a single interface to all Clients and Client versions.
- Requests contain a the list of Requested Attributes. Services send only Requested Attributes simplifying Client processing.
- The Schema contains pre-canned components which provide access to disparate data sources.

- The Schema creates a standardized Key:Value Response object which is returned to the Requestor.

Declarative interfaces promote a Plug and Play Infrastructure

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
Gaining and Evaluating Different Perspectives

Solution Entities are combined in any order for multiple perspectives.

Perspectives are then fed into the Evaluation Engine which produce Ranking Scores.

Fundamental Building Blocks

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
Observers Provide a Global View

Each Solution Entity Adapter implements an Observer Pattern which reports Events are Relevant State into a common Mapping Service Repository

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
Solution Entity States are preserved directly through their Adapters or the Observers into a high-performance graph structure which are easily be explored visually or through graph analytics.
The Problem To Solve

- Simplified mechanism for building and unifying disparate data sources into a normalize high performance Unified Data Fabric, which can be evaluated and understood. MITRE Grammars/Compiler provide a simple mechanism to create new graph structures to address emerging explorations through multiple services.
- Mechanisms to visually explore and understand both data entities and their relationships to one another.
  **Bloom**: Perspectives, Share Discoveries within teams, Near Natural Language processing
  **Browser**: Cypher developers interface simplified through our Grammars
- Mechanisms to understand and traverse threads of augmented entities. Templates and Solution Entities provide domain agnostic Rules Engine thread processing.
- Mechanisms to project and understand through graph analytics the desired dataset. Analytic Projections and Machine Learning integration through Data Science Library.

The Rules Simulation Engine is the Answer, One mechanism for all these things, and more.
Thank You

Ray Lukas (rlukas@mitre.org)
Jim Lockett (jdlockett@mitre.org)

The Attached Appendix Supplies More Detailed Explanations of RSE Grammars and Subgraphs

rlukas@MITRE.org

MITRE Labs inspires breakthroughs in applied science and advanced technology to transform the future of U.S. scientific and economic leadership. Our goal: Deliver disruptive innovation to support our mission of solving problems for a safer world.

Please feel free to contact me with any questions

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
### Database Comparisons

#### Database Overview

<table>
<thead>
<tr>
<th>Database</th>
<th>Purpose</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td>Table Based</td>
<td>Highly Refined</td>
</tr>
<tr>
<td>SQL Server</td>
<td>Rows and Columns</td>
<td>ACID Transactions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema Based</td>
</tr>
<tr>
<td>No SQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MongoDB</td>
<td>Key-Value Storage</td>
<td>Eventual Consistency</td>
</tr>
<tr>
<td>Cassandra</td>
<td>Document Storage</td>
<td></td>
</tr>
<tr>
<td>Spanner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anzo</td>
<td>Model Simplicity</td>
<td>White Board Friendly</td>
</tr>
<tr>
<td>Neo4J</td>
<td>High Speed Entity-Entity Traversals</td>
<td>ACID Transactions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schemaless</td>
</tr>
</tbody>
</table>

- *MongoDB contains ACID Transactions*
- *But Which Is The Best? It Depends On What You Are Doing*

**Relational**

Strict schema and data normalization separating data into tables. To preserve data consistency ACID transactions are supported. This imposes limitations on how relationships can be queried. Translations from OO to Rel are difficult and expensive.

- Relational model and other NoSQL database models link the data by implicit connections
- Relationships are reunified at query time

**Graph**

Whiteboard Friendly/Object Oriented: Native Graph databases have no pre-canned schema. Structures are directly mapped, any node can point to any other node. Unlimited query environment.

- Relationships are first-class citizen in a graph database and can be labelled, directed, and assigned properties
- Graphs and their Analytics Libraries are Very Scalable. Our graph technology can handle Trillions of nodes. Graphs with less than 10 million nodes and relationship are considered small.

**Performance:** Connections are made a creation time, not at query time

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
## Basic Graph Terminologies

**Grammars**: Grammars provide a simplified abstraction mechanism from the underlying graph technologies (in our case the Cypher Language) used to describe, and create graph, structures. Grammars are much like a high level programming language (C, C++, Java) which abstract the underlying CPU capabilities (assembly/machine code) mechanisms. *Grammars are compiled into the underlying Cypher codes.*

**Pattern**: Patterns are basic text strings (called ASCII Art) which describes a Node’s Relationship to another Node. (Node)-[Relationship]-(Node). *See Appendix for details.*

**Cypher**: Cypher is the language of Graphs. Our Grammars are abstractions of Cypher and are compiled into Cypher which can be easily deployed to create new high performant graph structures. Cypher’s compliment in the relational data world would be SQL. Cypher is basic *SQL for Graphs, but much better.*

**Create and Merge Commands**: Two basic and commonly used Cypher commands. Create creates the presented Pattern, even if it already exists. Merge uses the presented Pattern if it already exists, else it creates a new pattern in the database. *This is the underlying mechanism for auto connecting subgraphs and/or sharing threads in the Accumulation graphs.*

## Rules Simulation Engine Terminologies

**Rules Engine**: A Rules Engine is a service that allows domain (system) logic to be defined and invoked externally from the engine. Since the logic of the system is externalized from the engine, this allows the engine to be easily adapted to, and used across, virtually any domain. *This is a cornerstone concept in our solution, and how we provide a domain agnostic solution.*

**Simulation Path**: A graph is a series of interconnected entities (nodes). Our Rules Simulation Engine is able to traverse these pathways and for each node invoke externalized domain logic living inside autogenerated templates. This is how our Rules Simulation Engine implements a high performance, domain agnostic, decision/simulation mechanism. *This technique is generally used with Create and Accumulate (threading type) graphs. Merge graphs are generally for graph analytics.*
## RSE Grammar Node Types

### Standard Element Types

- **Node (instanceVar:LabelList{AttributeNameValueList})**
- **Relationship (node)-[instanceVar:RelationshipLabelList{AttributeNameValueList}]-{node}**

### Enhanced Element Types

#### Enhanced Visualization

**Index Nodes:** These are special autogenerated nodes which group like nodes together. For example, all the states in a country, all the counties in a state, all cities in a county all the data centers for each city. Each of these topology nodes are index nodes.

#### Graph Debugging

**Data Source Nodes:** Special nodes created for each data source that connect to all the nodes from that data source. Data Source nodes allow you to visualize the data sources for each node in a subgraph.

**Error Graphs:** ErrorType nodes are generated when errors are found in the underlying data source. Each ErrorType node points to nodes which describe each occurrence of such errors. Currently null data elements are supported, with additional error detection schemas under development.
Unified Data Fabric

**Subgraph Technology:** Each Data Source can define one or more subgraphs. Subgraphs can be loaded in any order, or not at all.

**Proxy Nodes:** Proxy Nodes (DS1:2 and DS3:1) are Placeholder (Proxy) nodes which define connection points that will be mapped to by other subgraphs. Proxy Nodes hold only the state information defined in that Data Source.

<table>
<thead>
<tr>
<th>Data Sources</th>
<th>Three Independent Sub Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Source 1</strong></td>
<td></td>
</tr>
<tr>
<td>DS1:1 [Atr1, Atr2, Atr3]</td>
<td></td>
</tr>
<tr>
<td>Item 1: Augmented [Atr1, Atr2, Atr3]</td>
<td></td>
</tr>
<tr>
<td>Item 2: Proxy Node</td>
<td></td>
</tr>
<tr>
<td><strong>Data Source 2</strong></td>
<td></td>
</tr>
<tr>
<td>DS2:2 [Atr4, Atr5]</td>
<td></td>
</tr>
<tr>
<td>Item 2: Augmented [Atr4, Atr5]</td>
<td></td>
</tr>
<tr>
<td>DS2:6 [Atr1, Atr3]</td>
<td></td>
</tr>
<tr>
<td>Item 6: Augmented [Atr1, Atr3]</td>
<td></td>
</tr>
<tr>
<td><strong>Data Source 3</strong></td>
<td></td>
</tr>
<tr>
<td>DS3:5 [Atr4, Atr1]</td>
<td></td>
</tr>
<tr>
<td>Item 5: Augmented [Atr4, Atr1]</td>
<td></td>
</tr>
<tr>
<td>DS3:4 [Atr1, Atr3]</td>
<td></td>
</tr>
<tr>
<td>Item 4: Augmented [Atr1, Atr3]</td>
<td></td>
</tr>
<tr>
<td>DS3:1 [Atr1, Atr3]</td>
<td></td>
</tr>
<tr>
<td>Item 1: Proxy Node</td>
<td></td>
</tr>
</tbody>
</table>

**Unified Data Set/Fabric**

<table>
<thead>
<tr>
<th>Load Order Agnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DS1:1</strong> [Atr1, Atr2, Atr3]</td>
</tr>
<tr>
<td>Item 2: Proxy Node</td>
</tr>
<tr>
<td><strong>DS2:2</strong> [Atr4, Atr5]</td>
</tr>
<tr>
<td>Item 6: Augmented [Atr1, Atr3]</td>
</tr>
<tr>
<td><strong>DS2:6</strong> [Atr1, Atr3]</td>
</tr>
<tr>
<td><strong>DS3:5</strong> [Atr4, Atr1]</td>
</tr>
<tr>
<td><strong>DS3:4</strong> [Atr1, Atr3]</td>
</tr>
</tbody>
</table>
| Subgraphs are Defined in the RSE V2.0 grammars. Load Only the Graphs/Data that you Need and Want for a Particular purpose.
Proxy definitions are similar to Foreign Keys in a Relational Database, but Hyper Performant.
Each Node Definition can be augmented with multiple property values.
Nodes are still able to invoke external templates and external resources.
External Templates can also modify graph structures.

**Proxy Node Mappings**

- DS1:2 is defined in DS2:2. When DS2 is loaded DS1:2 is mapped and augmented with Attribute 4 and Attribute 5
- DS3:1 is defined in DS1:1. Loading the Data Source 1 subgraph maps to DS3:1 and load Attribute 1, Attribute 2, and Attribute 3
## Visual Exploration Mechanisms

### Bloom
- Easily discover patterns yielding more questions and explorations.
- Near natural language visual explorations of graph structures without knowing SQL or Cypher. Bloom even can suggest entities to include in your visualizations.
- Provides mechanisms to easily update graph entities without knowing Cypher.
- Graph customizations (millions of colors, property-based styles, icons, auto sizing, etc.).
- Easily augmented capabilities through embedded Cypher enrichments.
- Provides defined perspectives which are tailored for a specific role or context.
- Subgraphs, Perspectives, Scenes, and other enrichments can be shared.

### GPU Accelerated Visualization
- High performance physics and rendering.

### Browser
- Developer Tool requires Cypher, the Language of Graphs, Knowledge.
- Limited display (colors, sizing of entities, no node icons, etc.) capabilities.

---

Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914
Node Definition

Data Source
Defines where this node should obtain its attributes. Current supported data sources include CSV files and Relational Database Tables.

Node Name
Node names are no longer restricted to the column name.

Identifier
List of the unique identifiers (primary key concept) for this node.

Augmentations
Defines additional attributes from the data source which should be included into this Node.

Improved Autogenerated Graph Structures
Accumulate, Create, and Merge graph

Grammar Components

<table>
<thead>
<tr>
<th>DS1</th>
<th>DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeName</td>
<td>DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})</td>
</tr>
<tr>
<td>Identifier</td>
<td>DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})</td>
</tr>
<tr>
<td>{Aug1, Aug2}</td>
<td>DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})</td>
</tr>
</tbody>
</table>
Subgraph Definition

- Each Data Source defines a subgraph. The DataCenter data source defines the following Data Center subgraph.
- Multiple subgraphs can, through our simplifies grammars, be defined each from separate or shared data sources and types.
- Subgraphs can be loaded in any order, or not at all. Loading a subgraph automatically connects like nodes together into a Master Graph.
- Only load the data you need for your specific explorations.
We can see that the Data Center Data Source contains two multifield elements, Assets and supported FISMA elements.

There are two types of Index nodes, both shown in Red. DataCenter Index node in the center points to all Data Center nodes. The Type Index node (Production, Validation, and Development) connect to their respective Data Center types.

Asset nodes are Proxy Nodes, containing on the Asset ID.
Asset Subgraph

Assets have a single Asset Index node pointing to all Assets
Assets have multiple Index nodes each describing various Asset characteristics
Vulnerability Subgraph

Vulnerability 003 and 002 are at High Risk
Vulnerability 001 and 002 require an Upgrade. 001 also requires a Reboot, etc.
The Master Graph contains all three subgraphs, and describes the Data Centers, Assets, and Vulnerabilities.

Assets (in Blue) are not connected to Vulnerabilities (in Gold) as shown in the Master Graph and the following Cypher Query.

```
neo4j$ match (a:Asset)-[]->(b:Vulnerability) return a, b
```

The underlying data sources have no such connections.
Merging Vulnerability and Assets

Standard Cypher, the language of graphs, as shown below, can be used to bind these entities together. The Bloom interface also provides this capability.

Merged Vulnerabilities and Assets

We can also see that some Assets, A330, B800, and B900 for example, are not bound in our Data Sets to a Data Center. This rogue assets might require further exploration.
Introduction to Cypher Patterns

Understanding Patterns

1. A Node has a Relationship to Another Node

2. \((\text{node}) - [\text{relationship}] - (\text{node})\)

3. \((:\text{Label \{Property\}}) - [\text{relationship}] - (:\text{Label \{Property\}})\)

4. \((e1:Employee\{name:"ray"\}) - [r1:worksWith] - (e2:Employee\{name:"steve"\})\)

5. \((e1:Employee\{name:"ray"\}) - [r1:worksWith] -> (e2:Employee\{name:"steve"\})\)

\((e1:Employee\{name:"ray"\}) - [r2:isAMemberOf] -> (g1:Group\{name:"research"\})\)
Generated Cypher Code

Load Nodes and Linkages Mechanism

```cypher
// auto USING PERIODIC COMMIT 300
load "file:///dataset.csv" as dataSetRow
with dataSetRow

ForEach(_ In Case When (dataSetRow.dataCenterColName is not Null)) Then [1] Else [] End

Merge (dataCenterInstance:DataCenter{name:"DataCenter",
dataCenterID:trim(dataSetRow.datacenter_id_derived)})
set dataCenterInstance.nType = trim("State")

ForEach(_ In Case When (dataSetRow.ServerID is not Null)) Then [1] Else [] End
Merge (serverInstance:Server{name:"Server", ServerID:trim(dataSetRow.ServerID)})
set serverInstance.macaddress = trim(dataSetRow.macAddress)
set serverInstance.dnsname = trim(dataSetRow.dnsname)
set serverInstance.nType = trim("State")
merge (dataCenterInstance)-[dataCenterToServerLink:contains]->(serverInstance)
// end For (dataSetRow.ServerID is not Null) command

// end For (dataSetRow.dataCenterColName is not Null) command
```
Generated Cypher Code

Multifield Mechanism

:auto USING PERIODIC COMMIT 300  load csv with headers from "file:///newVersion.csv" as osRow with osRow

1. ForEach(_ In Case When ((osRow.ServerID Is not Null)) Then [1] Else [] End|
   Merge (serverInstance:Server{name:"Server", ServerID:trim(dataSetRow.ServerID)})

2. ForEach(_ In Case When (osRow.os is not null) Then [1] Else [] End|

3. ForEach(osItem in split(osRow.os, ';') Then [1] Else [] End|

4. ForEach(_ IN CASE WHEN (osItem <> '') Then [1] Else [] End|

   merge (osInstance:OS{name:"OS", operSys:trim(osItem)})
   set osInstance.nType = trim("Index")
   merge (serverInstance)-[:isA]->(osInstance)

5. )

)
MITRE Labs inspires breakthroughs in applied science and advanced technology to transform the future of U.S. scientific and economic leadership. Our goal: Deliver disruptive innovation to support our mission of solving problems for a safer world.

Please feel free to contact me with any questions

Thank You

Ray Lukas (rlukas@mitre.org)
Jim Lockett (jdockett@mitre.org)

rlukas@MITRE.org