Rule-Based Engineering Using Declarative Graph Database Queries

Sten Grüner, Ulrich Epple
Chair of Process Control Engineering, RWTH Aachen University

MBEES 2014, Dagstuhl, 05.03.14
Motivation

- Every plant is a highly customized system build up from stock components according to the customer’s requirements.

- Commissioning and reconfiguration of chemical and manufacturing plants requires a high amount of engineering = creation and testing of control structures from a pre-defined set of components.
  - Done manually → high costs

- Consequents: engineering costs become variable for re-configurable plants and hamper the uptake of innovations.

- Goal of Industrie 4.0: coupling of production facilities owned by different organizations.
Motivation

A possible solution is the utilization of a rule-based system

Pros:
- “Separation of concerns”
- A rule base is (ideally) independent of the current project
- Helps to overcome error prone “copy paste” tasks

Cons:
- An additional complex system
- Where does the rule-base come from?
- Overfitting of the rules

Our proposal: define rules using declarative graph-based queries
Talk Overview

- Applications of rule-based systems in industrial automation

- Technology overview
  - P&I diagram data
  - ACPLT automation server
  - Neo4J and Cypher

- Graph data rule-based system

- Use-cases
  - Code generation out of P&I diagrams
  - Flow path discovery

- Summary and outlook
Rule-based Engineering Systems – An Overview

- The idea of rule-based systems is not new
  - An organizational paradigm of storing engineering knowledge
  - Original ideas come from knowledge-based domain
  - First applications in the automation domain around 1980th

- Part of a broader research field “Automation of Automation”
  - Asset monitoring
  - Assistance systems
  - Human-Machine-Interface

- **Core design decision**: how are the rules represented and evaluated
  - Typically in a permission-conclusion (IF-THEN) format
Main application area of RBES: model transformations
- Generate one model based on other model(s)

Two studied use-cases from the publication by Kraußer at al.:
1. Generate basic control blocks based on plant description:
   - E.g. create a motor control block in the PLC for each pump on the plant plan
2. Model exploration:
   - E.g. follow pipes between any two vessels to find a possible flow path that can be monitored

Further application areas:
- HMI generation
- I/O configuration generation
- Etc.
Talk Overview

- Applications of rule-based systems in industrial automation

- Technology overview
  - P&I diagram data
  - ACPLT automation server
  - Neo4J and Cypher

- Graph data rule-based system

- Use-cases
  - Code generation out of P&I diagrams
  - Flow path discovery

- Summary and outlook
P&I Diagrams

- Piping and instrumentation diagrams – core model for plant structure:

- Plant elements and PCE requests
- Standardized in IEC 62424
PandIX

- Piping and Instrumentation Exchange:
  - CAEX profile for describing P&IDs
  - “XML-Format for P&IDs”

- How to obtain PandIX data?
  - Manual creation
  - Export of CAD/CAE tools e.g. Siemens COMOS or Smartplant

- Core concept:
  - Elements with roles and connection ports
  - Directed connections of different types
  - XML hierarchy is used to model hierarchical element composition
PandIX – Example

[...]  
<InternalElement Name="N1">  
  <ExternalInterface Name="PIn"  
    RefBaseClassPath="ProductConnectionPoint"/>  
  <ExternalInterface Name="POut"  
    RefBaseClassPath="ProductConnectionPoint"/>  
  <ExternalInterface Name="N"  
    RefBaseClassPath="ActuationPoint"/>  
  <RoleRequirements  
    RefBaseRoleClassPath="PumpRequest"/>  
</InternalElement>

[...]  
<InternalLink Name="N1-Y1"  
  RefPartnerSideA="N1:POut"  
  RefPartnerSideB="Y1:PIn"/>  
[...]
ACPLT Automation Server

- Portable ANSI-C implementation of a runtime environment

- Function blocks as main programming paradigm
  - OO-Features e.g. inheritance and polymorphism

- Message and signal-oriented inter-block communication

- Reconfiguration at runtime

- Binary and HTTP RESTful interfaces (OPC-UA in development)

- Open Source
Function Block Diagram

- A graphical programming language from IEC 61131
- Composition of function blocks
- Function block’s interface from the outside
- Implementation available in the ACPLT-environment
The presented models (P&IDs and FBDs) have a native graph representation.

Neo4J and Cypher treat nodes and edges as first-class citizens.

Cypher graph patterns are intuitively understandable:
- No SQL knowledge required
- A leap from ASCII representation to graphical query representation is possible
- Can work as a bridge between computer scientists and automation engineers
Neo4J

- Open-Source noSQL Graph-Based-Database
  - can be faster than typical SQL engines
  - Implements a graph query language Cypher

- Data Model: attributed multi-graph
  - Nodes, Pointed relations of different types
  - Both have a key-value attribute table
Cypher

- Cypher
  - SQL-similar language for declarative graph queries

- Example for a read-only query:
  - START n=node(*)
    MATCH n-[[:participating]->()<-[:participating]]-m
    RETURN n.name, m.name;

- Returns a table:

<table>
<thead>
<tr>
<th>n.name</th>
<th>m.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grüner</td>
<td>Kampert</td>
</tr>
<tr>
<td>Kampert</td>
<td>Grüner</td>
</tr>
</tbody>
</table>
Talk Overview

- Applications of rule-based systems in industrial automation

- Technology overview
  - P&I diagram data
  - ACPLT automation server
  - Neo4J and Cypher

- Graph data rule-based system

- Use-cases
  - Code generation out of P&I diagrams
  - Flow path discovery

- Conclusion and outlook
Architecture of Graph Data Rule-Based System

Rule set
\[\text{Rule engine}\]
\[\text{Neo4J}\]
\[\text{Rule-based engineering system}\]

Extension of the rule set
Modification using operation calls
Import into DB

Tools containing engineering data
Eng. tool A  Eng. tool B  Runtime environment

Engineering data in different formats
P&ID  P&ID  CAD  Logic
PandIX  Proprietary  AutomationML  PLC-Open
XML

Slide 17/31
S. Grüner, U. Epple: Rule-Based Engineering Using Declarative Graph Database Queries
Rule Representation and Evaluation

- IF-THEN logic replaced by FORALL-DO logic in RuleML-like syntax:

```xml
<rule>
  <forall>
    <query><![CDATA[
      START n=node(*) MATCH n-[[:participating]]->()<-[:participating]-m RETURN n.name, m.name ]]]></query>
  </forall>
  <op>System.out.println("{n.name} and {m.name} participate at the same event")</op>
</rule>
```

- Operations are evaluated on the rows of the table
- Tokens matching column names are replaced in the operation calls
- Java reflection used to keep the system easily extendable
Implemented Operations

- Text based I/O
  - Console output
  - Writing into files

- KS / HTTP protocol wrapper to communicate to ACPLT servers
  - Creating function blocks / connection
  - Reading variables

- Neo4J API
  - Creating properties for nodes
  - Creating links between nodes
Talk Overview

- Applications of rule-based systems in industrial automation

- Technology overview
  - P&I diagram data
  - ACPLT automation server
  - Neo4J and Cypher

- Graph data rule-based system

- Use-cases
  - Code generation out of P&I diagrams
  - Flow path discovery

- Summary and outlook
Use-Case Architecture

Rule-Based Engineering Using Declarative Graph Database Queries

Tools containing engineering data

Runtime environment

Modification using operation calls

Import into DB

Engineering data in different formats

Rule set

Rule engine

DB

Neo4J

P&ID

PandIX

Rule-based engineering system
Use-Case 1:

- Generation of pre-defined single control block instances:
Use-Case 1: Imported Graph Data

Slide 23/31
Use-Case 1: Rule

```
<rule>
  <forall>
    <query><![CDATA[
      START n=node(*)
      MATCH pump-[[:CONTAINMENT]]->()
        -[:ACTUATION_CONNECTION]-()<-[:CONTAINMENT]-pce
      WHERE pump.role = "PumpRequest"
      AND pce.role = "ActuatorRequest"
      AND pce.functionCode = "N"
      RETURN distinct id(pump), pump.Name, pce.Name, pce.SignalCode
    ]]>]
    <![CDATA[
      <op>fn.KS.createObject("Pump{pce.SignalCode}/TechUnits", "{pce.Name}")</op>
      <op>fn.Neo4j.createPropForNodeid("id(pump)", "basicControl", "/TechUnits/{pce.Name}")</op>
    ]]>]
  </forall>
</rule>
```
Use-Case 2:

Possible flow paths between any pair of vessels
Use-Case 2: Flow Types

- Internal and external flows occur only between **product ports**:
  - External flows = flow in pipes – both directions
  - Internal flows = flow inside of an element – type dependent

- Assumptions:
  - Pumps allow flows in one directions
  - Fittings and valves allow flows in any direction
Use-Case 2: Rule Query

START n=node:node_auto_index(Name="plant")
MATCH v1-[[:CONTAINMENT]]->p1,
    p2<-[:CONTAINMENT]-v2,
    path=p1-[:INTERNAL_FLOW|EXTERNAL_FLOW*..16]->p2
WHERE v1.role = "VesselRequest"
    AND v2.role = "VesselRequest"
    AND ALL(n IN nodes(path) WHERE
        length(filter(m IN nodes(path)
            WHERE m.parent=n.parent)
        )<=2
    )
RETURN DISTINCT extract(x in nodes(path) |
    [x.parent, x.Name])
Use-Case 2: Evaluation Result

[["V1", "POutBottom"], ["Z1", "P1"], ["Z1", "P3"],
 ["Y1", "PIn"], ["Y1", "POut"], ["Z2", "P3"],
 ["Z2", "P2"], ["V2", "POutTop"]]

[["V1", "POutBottom"], ["Z1", "P1"], ["Z1", "P3"],
 ["N1", "PIn"], ["N1", "POut"], ["Z2", "P1"],
 ["Z2", "P2"], ["V2", "POutTop"]]

[["V2", "POutTop"], ["Z2", "P2"], ["Z2", "P3"],
 ["Y1", "POut"], ["Y1", "PIn"], ["Z1", "P3"],
 ["Z1", "P1"], ["V1", "POutBottom"]]

[["V2", "POutBottom"], ["N2", "PIn"],
 ["N2", "POut"], ["V1", "POutTop"]]
Use-Case 2: Drawbacks

- Problem 1: The query is not intuitively understandable

- Problem 2: Performance problems
  - The execution of the query takes about 23 minutes [sic]
  - The problem is the query evaluation mechanism:
    - All the paths up to length 16 are accumulated at first
    - Match filter is applied on the accumulated path set
    - Number of paths grows exponentially
    - Execution time grows exponentially

- Lessons learned:
  - Cypher syntax/evaluation needs to be improved to formulate path properties and filter them on the fly
  - **Strength of graph based rules**: local, pattern-matching applications
Talk Overview

- Applications of rule-based systems in industrial automation

- Technology overview
  - P&I diagram data
  - ACPLT automation server
  - Neo4J and Cypher

- Graph data rule-based system

- Use-cases
  - Code generation out of P&I diagrams
  - Flow path discovery

- Summary and outlook
Summary and Outlook

- We presented a graph query rule-based engineering system

- The system relies on a graph database and graph queries
  - Rule description formalism is accessible to both engineers and computer scientists

- The application of the system is neither limited to industrial automation engineering nor to ACPLT technologies

- Further work: extension and verification of the system to new application areas
  - Further import plugins
  - Further operation implementations
Rule Set

- Due to the simplicity of the engine the know-how is stored in the rule set

- Options of rule-set creation:
  - Manual design for a particular task
    +: good results, simple
    -: expensive
  
  - Automatic generation from existing data using machine learning
    +: less overhead
    -: expert insight still needed
  
  - Assistance system
    +: flexible, on the fly creation and testing
Function Blocks

- Function block is the smallest program organization unit

- Known concept from IEC 61131-3, IEC 61499 and MATLAB/Simulink

- Data encapsulation through clear interfaces

- Native Blocks: algorithm implemented in a higher language (C, ST, etc.) (black-box)

- Aggregated Blocks: internal logic composed from further function blocks (white-box)