McGill

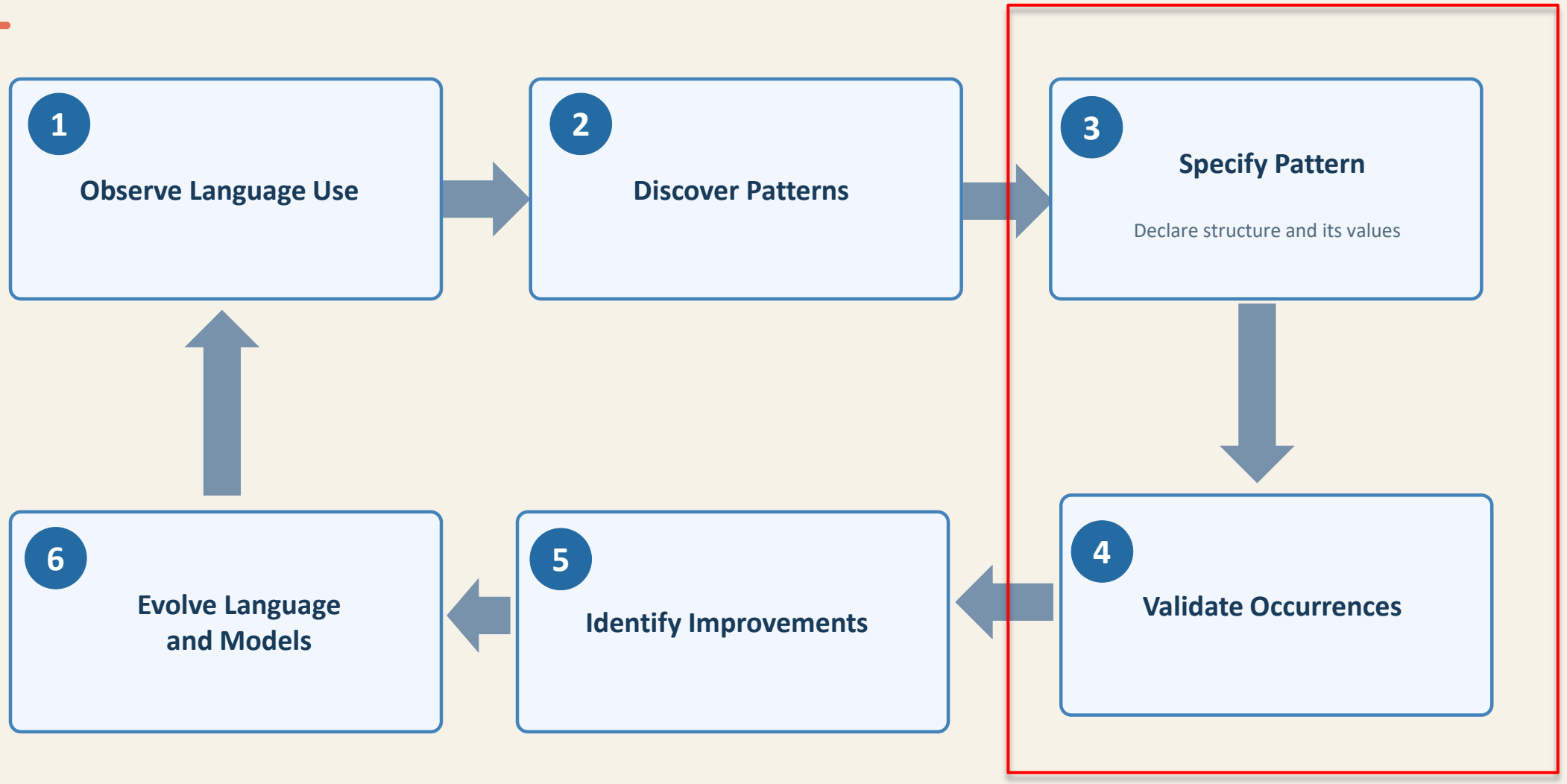
A Domain Specific Language for Pattern Specification and Automated Validation for Metamodel-Based Languages

Ankita Vyas · Gunter Mussbacher

McGill University · Montréal · August 17, 2026

MoDRE'26

Motivation



Feedback loop for Self-Adaptable Languages

Challenges in Structural Pattern Validation

Validating recurring structures over EMF models presents two challenges:

- **Sequence-sensitive relations.**
- **From discovery to executable validation.**

```
1 {job_name}:
2     needs:    {value}
3     runs-on:  {value}
4     name:     {value}
5     if:       {value}
6     steps:    {value}
```

YAML Pattern: A recurring job-level structural pattern in GitHub Actions workflows

Existing Approach: OCL for EMF Model Validation

```
context Pipeline                                                    1
def: hasPatternOccurrence(): Boolean =                              2
self.jobs->exists(j |                                               3
    not j.name.oclIsUndefined() and                                 4
    not j.dependKW.oclIsUndefined() and                             5
    j.steps->notEmpty() and                                         6
    let params : Sequence(ParameterValue) =                         7
    j.jobParameterValues->asSequence() in                           8
    params->exists(pRun |                                           9
        pRun.name = PipelineKeyword::PPL_KW_RUNS_ON and             10
        params->exists(pName |                                      11
            pName.name = PipelineKeyword::PPL_KW_NAME and           12
            params->indexOf(pName) > params->indexOf(pRun) and      13
            params->exists(pIf |                                    14
                pIf.name = PipelineKeyword::PPL_KW_IF and           15
                params->indexOf(pIf) > params->indexOf(pName)       16
            )                                                       17
        )                                                           18
    )                                                               19
)                                                                   20
```

OCL Query for YAML Pattern with Ordered Criteria

Proposed Approach

One compact specification becomes a repeatable corpus-level check.

Explicit relation-level ordering • Generated standalone java validators • Filename-level results

The Same Ordered YAML Pattern in OCL and PDL

```
context Pipeline
def: hasPatternOccurrence(): Boolean =
self.jobs->exists(j |
    not j.name.oclIsUndefined() and
    not j.dependKW.oclIsUndefined() and
    j.steps->notEmpty() and
    let params : Sequence(ParameterValue) =
    j.jobParameterValues->asSequence() in
    params->exists(pRun |
        pRun.name = PipelineKeyword::PPL_KW_RUNS_ON and
        params->exists(pName |
            pName.name = PipelineKeyword::PPL_KW_NAME and
            params->indexOf(pName) > params->indexOf(pRun) and
            params->exists(pIf |
                pIf.name = PipelineKeyword::PPL_KW_IF and
                params->indexOf(pIf) > params->indexOf(pName)
            )
        )
    )
)
```

Ordered YAML pattern in OCL

```
Project_Path: "ca.mcgill.devops.pipeline"
Pipeline {
    (Job)jobs* {
        {
            (String)name,
            (String)dependKW,
            (ParameterValue)jobParameterValues* Ordered {
                { (PipelineKeyword)name: "runs-on:" },
                { (PipelineKeyword)name: "name:" },
                { (PipelineKeyword)name: "if:" }
            },
            (Step)steps* { }
        }
    }
}
```

Ordered YAML pattern in PDL

Generation of Java Validators – Acceleo Transformation


```
Project_Path : "ca.mcgill.devops.pipeline"
// Root Class match starts here
Pipeline {
  (Job)jobs* {
      {
         (String)name,
         (String)dependKW,

         (ParameterValue)jobParameterValues*
Ordered {
            {
               (PipelineKeyword)name: "name:"
            },
            {
               (PipelineKeyword)name: "runs-on:"
            },
            {
               (PipelineKeyword)name: "if:"
            }
         },
         (Step)steps* {
         }
      }
   }
}
```

1

```
//Use of ProjectPath for relevant Imports
package ca.mcgill.devops.pipeline.resourceloader;
....
import ca.mcgill.devops.pipeline.PipelineStandaloneSetup;
import ca.mcgill.devops.pipeline.pipeline.*;
```

2

```
//Root Class match
private static boolean matchesPattern(EObject root) {
   if (root instanceof Pipeline) {
      return matchesPatternPipeline((Pipeline) root);
   }
   return false;
}

private static boolean matchesPatternPipeline(Pipeline root) {
   return matchesJobJobs_1(root);
}
```

Transformation Example for Ordered YAML Pattern Model -> Java Validation Code

Generation of Java Validators – Acceleo Transformation


```
Project_Path : "ca.mcgill.devops.pipeline"
// Root Class match starts here
Pipeline {
 (Job)jobs* {
     {
        (String)name,
        (String)dependKW,

        (ParameterValue)jobParameterValues*
Ordered {
           {
              (PipelineKeyword)name: "name:"
           },
           {
              (PipelineKeyword)name: "runs-on:"
           },
           {
              (PipelineKeyword)name: "if:"
           }
        },
        (Step)steps* {
        }
     }
  }
}
```

3

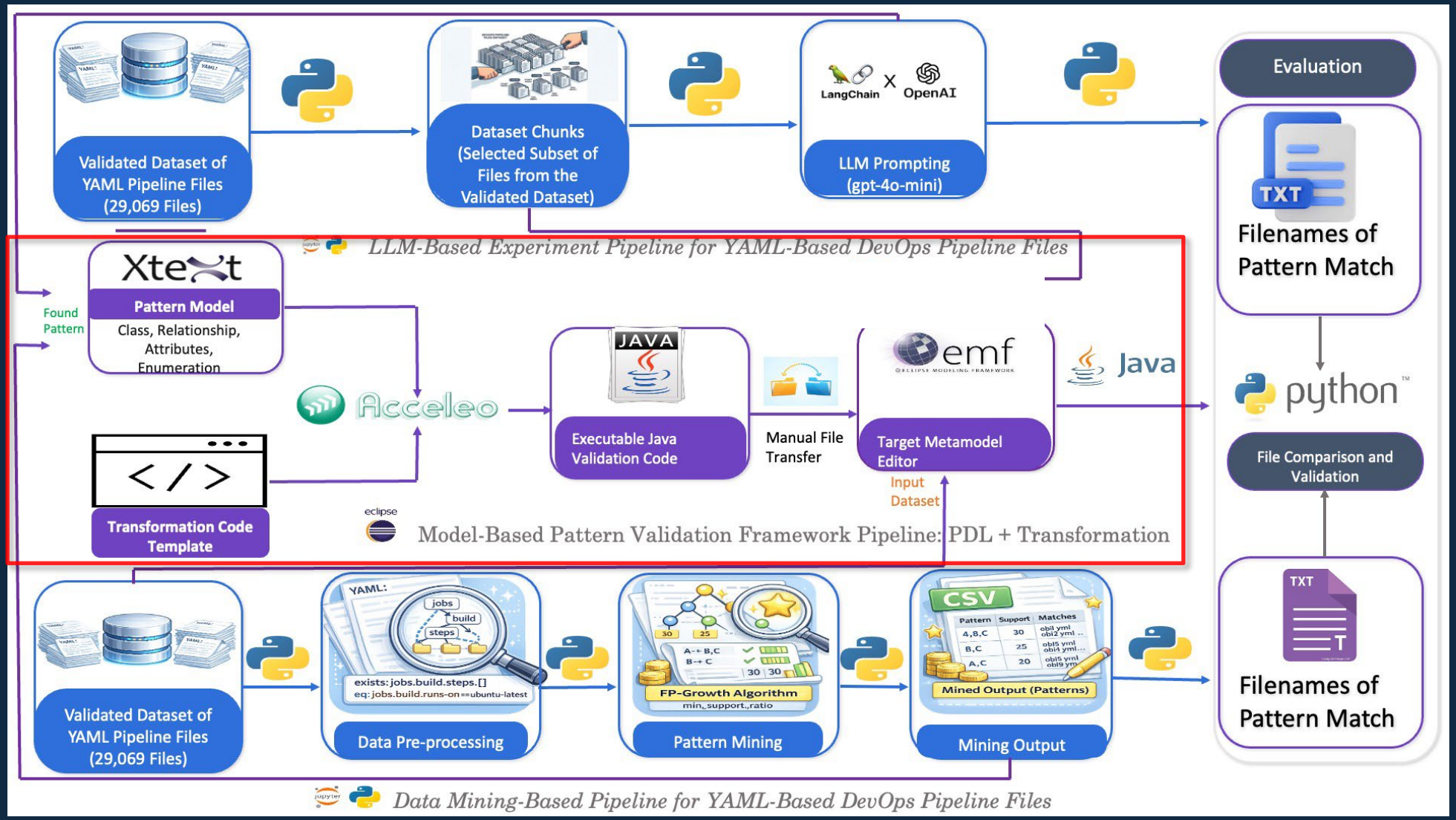
```
//Non – Empty Relation match
private static boolean matchesJobJobs_1(Pipeline classElem) {

   EStructuralFeature f =
classElem.eClass().getEStructuralFeature("jobs");
   Object raw = classElem.eGet(f);
   EList<?> list = (EList<?>) raw;

   for (Object item : list) {
      if (!(item instanceof Job)) continue;
      Job elem = (Job) item;
      // perform Attribute / Enumeration / Relation match here
as needed
      ….
   }
   return true or false;
}
```

Transformation Example for Ordered YAML Pattern Model -> Java Validation Code

Evaluation Pipeline



Evaluation Results – Data Mining Approach

```
1  name : {value}
2  on:
3      push:
4          branches:
5              {value}
6
```

Pattern Found by Data Mining Approach

6,788 common; 3 mining-only;
2 Java-only

Evaluation Results – LLM Approach

```
1  name:
2  on:
3      push:
4          branches:
5      pull_request:
6  jobs:
7      build:
8          runs-on:
9          steps:
10             - uses:
11             - name:
12                 run:
13             - name:
14                 run:
```

Pattern Found by LLM Approach

Ordered

Exact match count with Python Reference Matcher

Interpretation of Results

11

patterns

8 unordered •
3 ordered

- **Realistic patterns from actual GitHub Actions - DevOps pipelines.**
- **Consistent filename-level agreement.**
- **Representation explains mismatches.**
- **Ordering increases selectivity.**
- **Traceable mismatch analysis.**
- **Evidence for metamodel evolution.**

Limitations and Future Work

LIMITATIONS

- Evaluation covers one use-case metamodel i.e., the DevOps Pipeline metamodel.
- Manual PDL reconstruction requires knowledge of the target metamodel.
- LLM evaluation was limited to selected dataset subsets due to GPT-4o-mini's context limit.
- FP-Growth identifies structurally co-occurring features without considering their order.

FUTURE WORK

- Evaluate PDL across additional metamodels and application domains.
- Automate or Reduce the manual effort required to reconstruct discovered patterns in PDL.
- Explore alternative LLM models, stronger prompting and grounding techniques with larger dataset coverage.
- Investigate other data-mining algorithms capable of discovering ordered structural patterns.

Conclusion

Pattern Specification

Allows nested, repeated, value-constrained, unordered, and ordered patterns through Xtext-based PDL.

Validator Generation

Generates a pattern-specific Java validator through Acceleo.

Empirical Validation

Most validators achieved exact filename-level agreement with the reference workflows.

Language Evidence

Reveals recurring structures and differences between textual and metamodel representations.