ETRI OMiLAB®

SAC 2025 - Programming Language

A Platform-Independent Software-Intensive Workflow Modeling Language And An Open-Source Visual Programming Tool

A Bottom-Up Approach Using Ontology Integration Of Industrial Workflow Engines

Dr. Yong-Jun Shin, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea
yjshin@etri.re.kr

Dr. Wilfrid Utz, OMiLAB NPO, Berlin, Germany
Wilfrid.utz@omilab.org

April 3, 2025

Outline

1. Introduction
2. Workflow modeling language and Tool
3. Evaluation
4. Conclusion

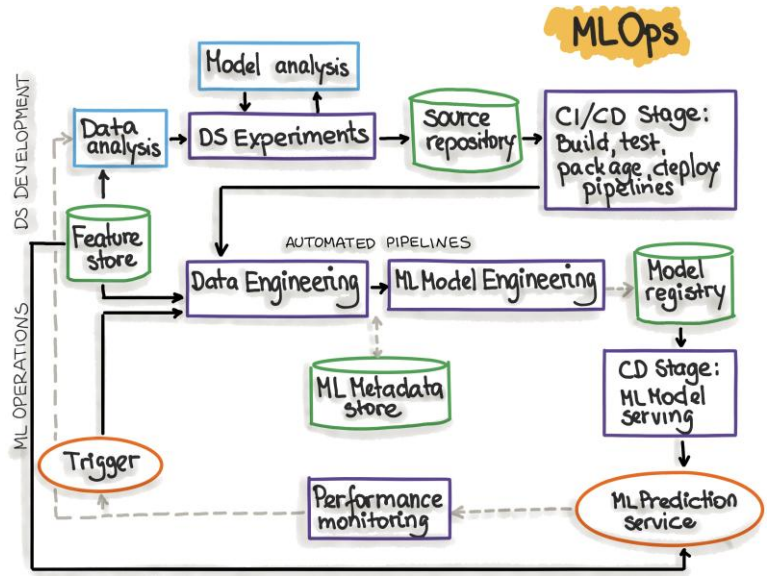
A Platform-Independent Software-Intensive Workflow Modeling Language And An Open-Source Visual Programming Tool

Introduction

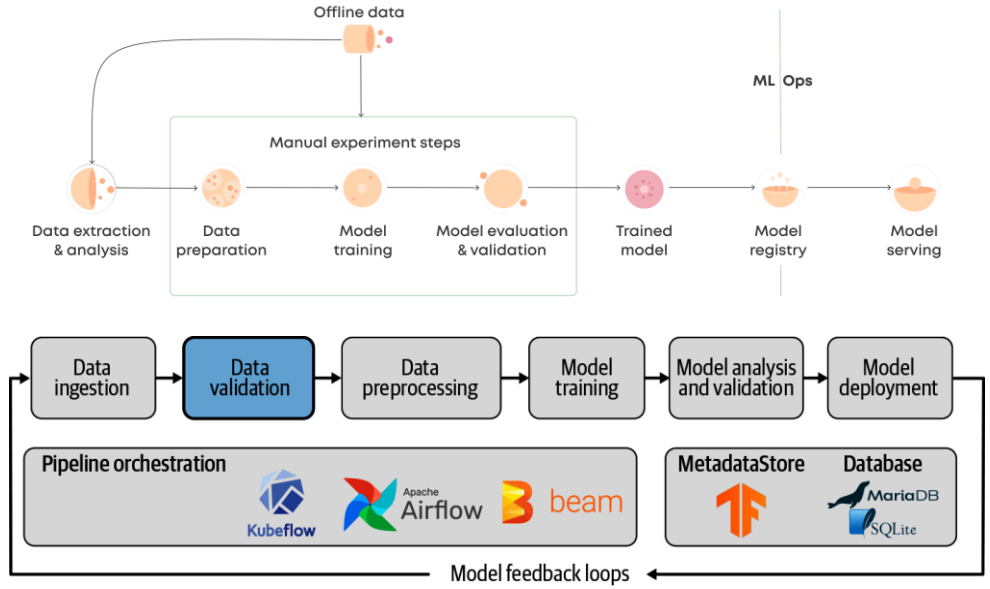
- Workflow engines and workflow software specifications
- Necessity of platform-independent visual language for workflow software

Software-intensive Workflows

- Workflow is an effective tool to decompose and manage complex services (e.g., AI-enabled services, MLOps)
  - Automated execution, periodic execution, auto-repair, regular report, etc.



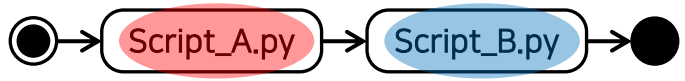
Cooperating tasks
using heterogeneous platforms, libraries, etc.



Orchestrating workflow of independent tasks

Workflow Specification of Industrial Workflow Engines

- Many industrial platforms provides code-based specification and automated execution of workflow services based on their own grammars.



argo

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: sequential-pipeline-
spec:
  entrypoint: sequential-steps
  templates:
  - name: sequential-steps
    steps:
    - - name: step-a
        template: run-script
        arguments:
          parameters:
          - name: script
            value: "script_A.py"
    - - name: step-b
        template: run-script
        arguments:
          parameters:
          - name: script
            value: "script_B.py"

  - name: run-script
    inputs:
      parameters:
      - name: script
    container:
      image: python:3.9
      command: ["python"]
      args: ["{{inputs.parameters.script}}"]
```

Apache Airflow

```
from airflow import DAG
from airflow.operators.bash import BashOperat
or
from datetime import datetime

default_args = {
    'start_date': datetime(2024, 3, 11),
    'catchup': False
}

with DAG('sequential_workflow', default_args=
default_args, schedule_interval=None) as dag:

    step_a = BashOperator(
        task_id='run_script_A',
        bash_command='python script_A.py'
    )

    step_b = BashOperator(
        task_id='run_script_B',
        bash_command='python script_B.py'
    )

    step_a >> step_b  # Defines the execution
order
```

Kubeflow

```
from kfp import dsl

@dsl.pipeline(
    name='Sequential Workflow',
    description='A pipeline that runs two scr
ipts sequentially.'
)
def sequential_pipeline():
    step_a = dsl.ContainerOp(
        name='Run Script A',
        image='python:3.9',
        command=['python'],
        arguments=['script_A.py']
    )

    step_b = dsl.ContainerOp(
        name='Run Script B',
        image='python:3.9',
        command=['python'],
        arguments=['script_B.py']
    )

    step_a >> step_b  # Defines the execution
order

if __name__ == '__main__':
    import kfp.compiler as compiler
    compiler.Compiler().compile(sequential_pi
peline, 'sequential_pipeline.yaml')
```

METAFLOW

```
from metaflow import FlowSpec, step

class SequentialWorkflow(FlowSpec):

    @step
    def start(self):
        print("Running script_A.py")
        import os
        os.system("python script_A.py")
        self.next(self.step_b)

    @step
    def step_b(self):
        print("Running script_B.py")
        import os
        os.system("python script_B.py")
        self.next(self.end)

    @step
    def end(self):
        print("Workflow completed.")

if __name__ == "__main__":
    SequentialWorkflow()
```

Necessity of Platform-independent WorkflowML

- Challenges
  - Inefficient code-based workflow specification (e.g., hundreds of lines)
    - Error-prone process and poor communicability
  - Platform-independent grammars
    - Difficult platform migration despite common semantics

- Goal
  - Platform-independent and visual **language** for workflow specification
  - Open-source workflow modeling **tool**

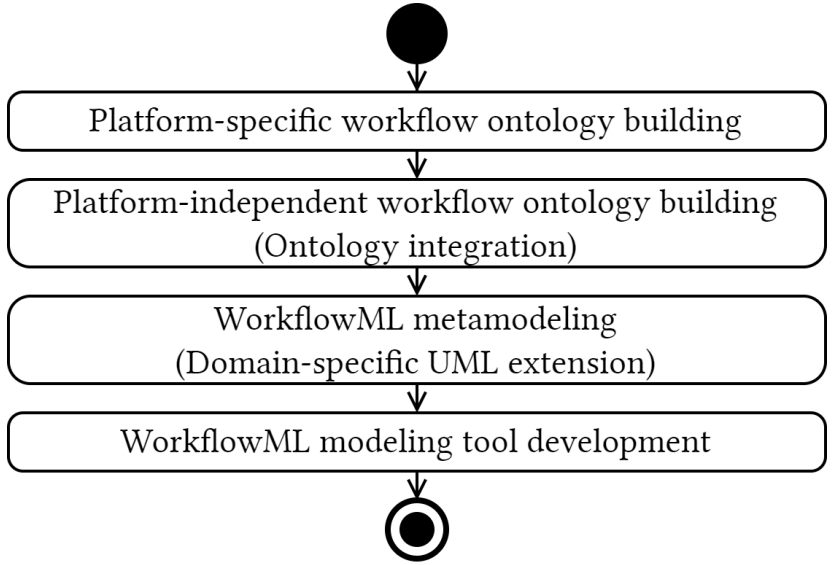
A Platform-Independent Software-Intensive Workflow Modeling Language And An Open-Source Visual Programming Tool

Workflow modeling language (WorkflowML) and Tool

- WorkflowML metamodeling
- WorkflowML tool development on ADOxx

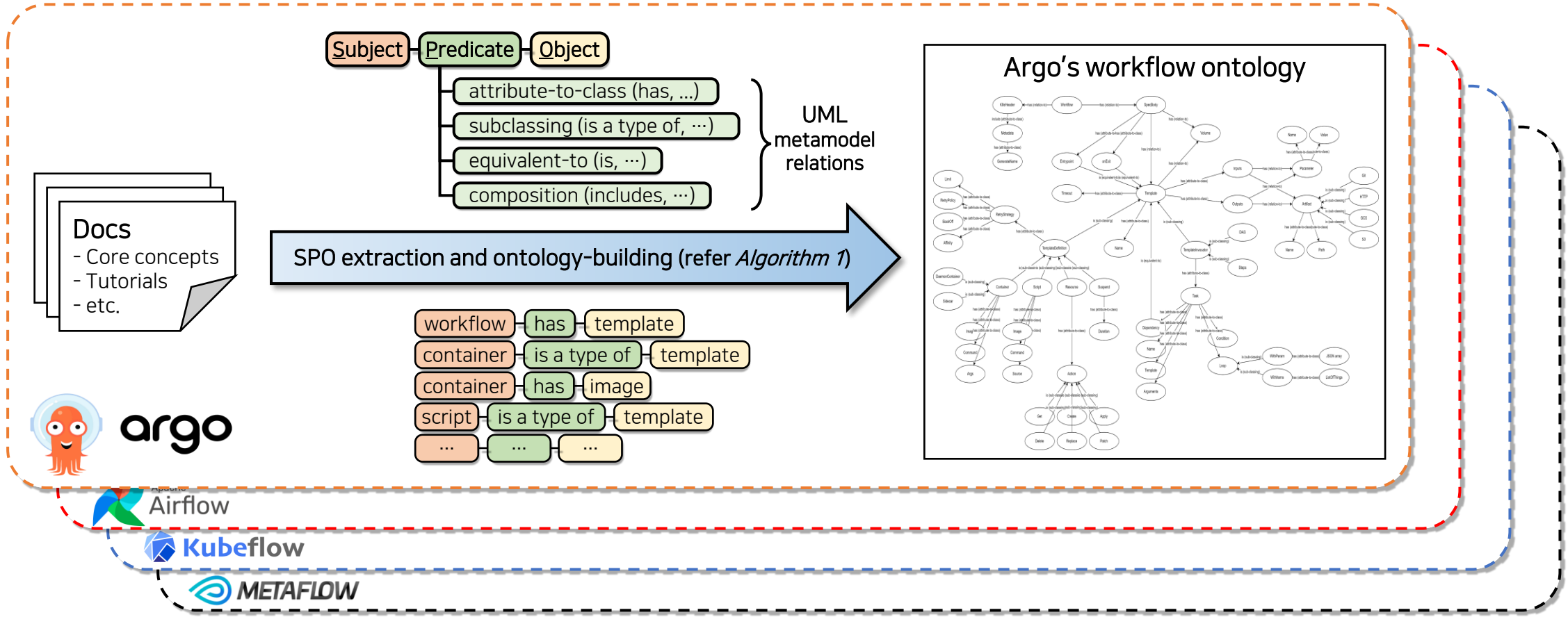
Overall Approach

- Bottom-up WorkflowML development
  - Collect workflow concepts from platforms
  - Integrate workflow concepts
  - Develop WorkflowML extending UML activity diagram
- ADOxx-powered visual programming tool
  - https://adoxx.org/
  - Open-use metamodeling platform of OMiLAB NPO
  - Easy development and deployment of domain-specific modeling language



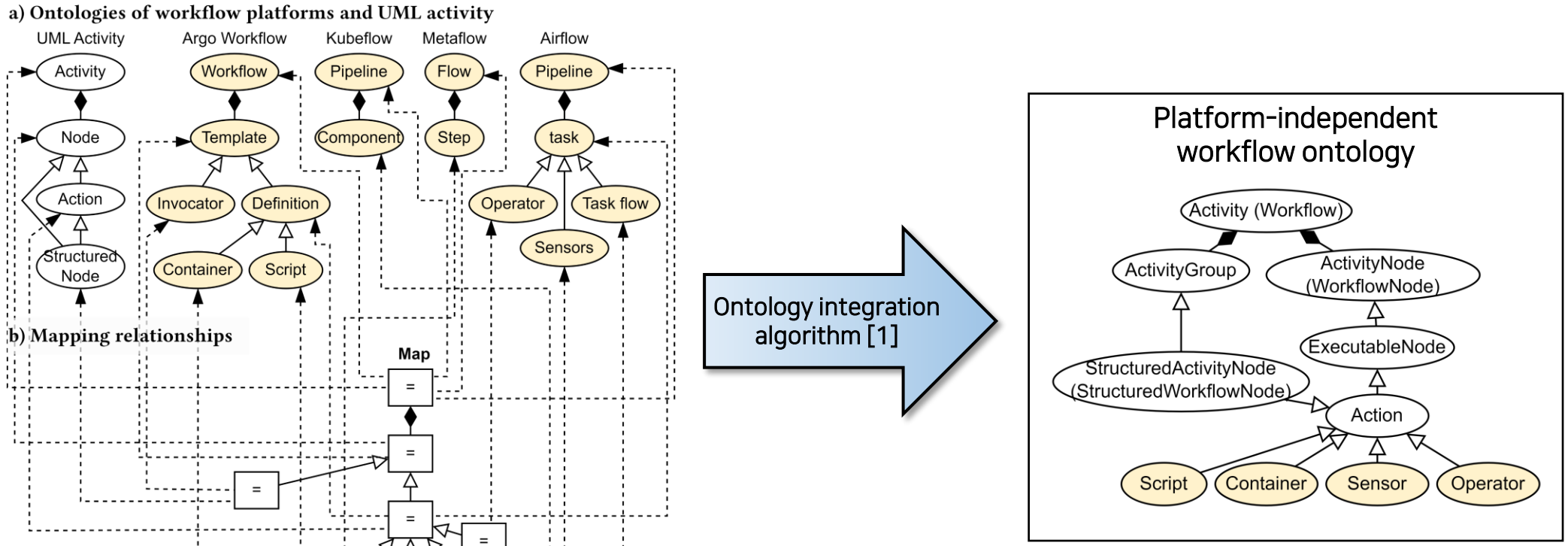
Platform-specific Ontology Building

- Collect ontologies (graphs) from independent workflow platforms.



Platform-Independent Ontology Building (1/2)

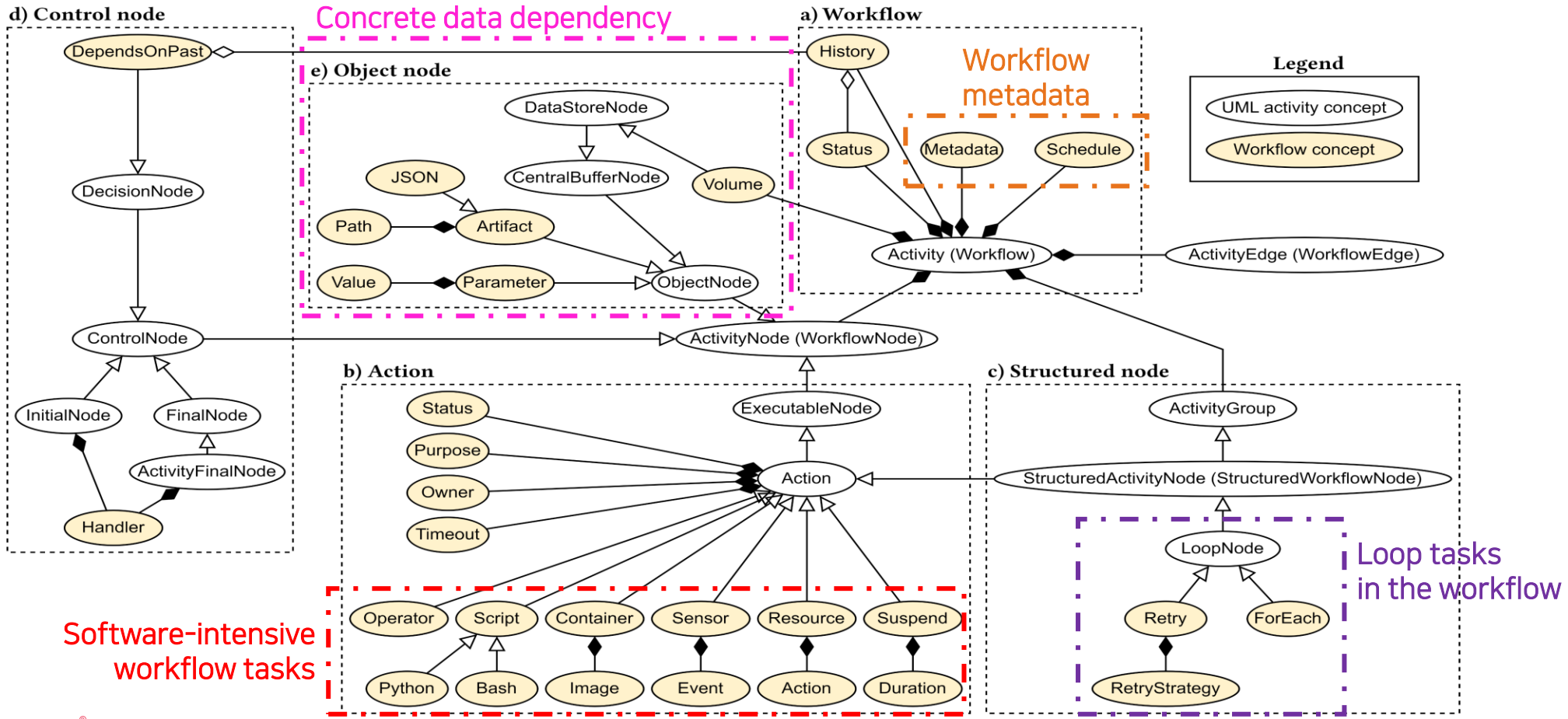
- Integrate ontologies of platform-specific workflows and UML activity



[1] Rachel A Pottinger and Philip A Bernstein. 2003. Merging models based on given correspondences. In Proceedings 2003 VLDB Conference. Elsevier, 862–873.

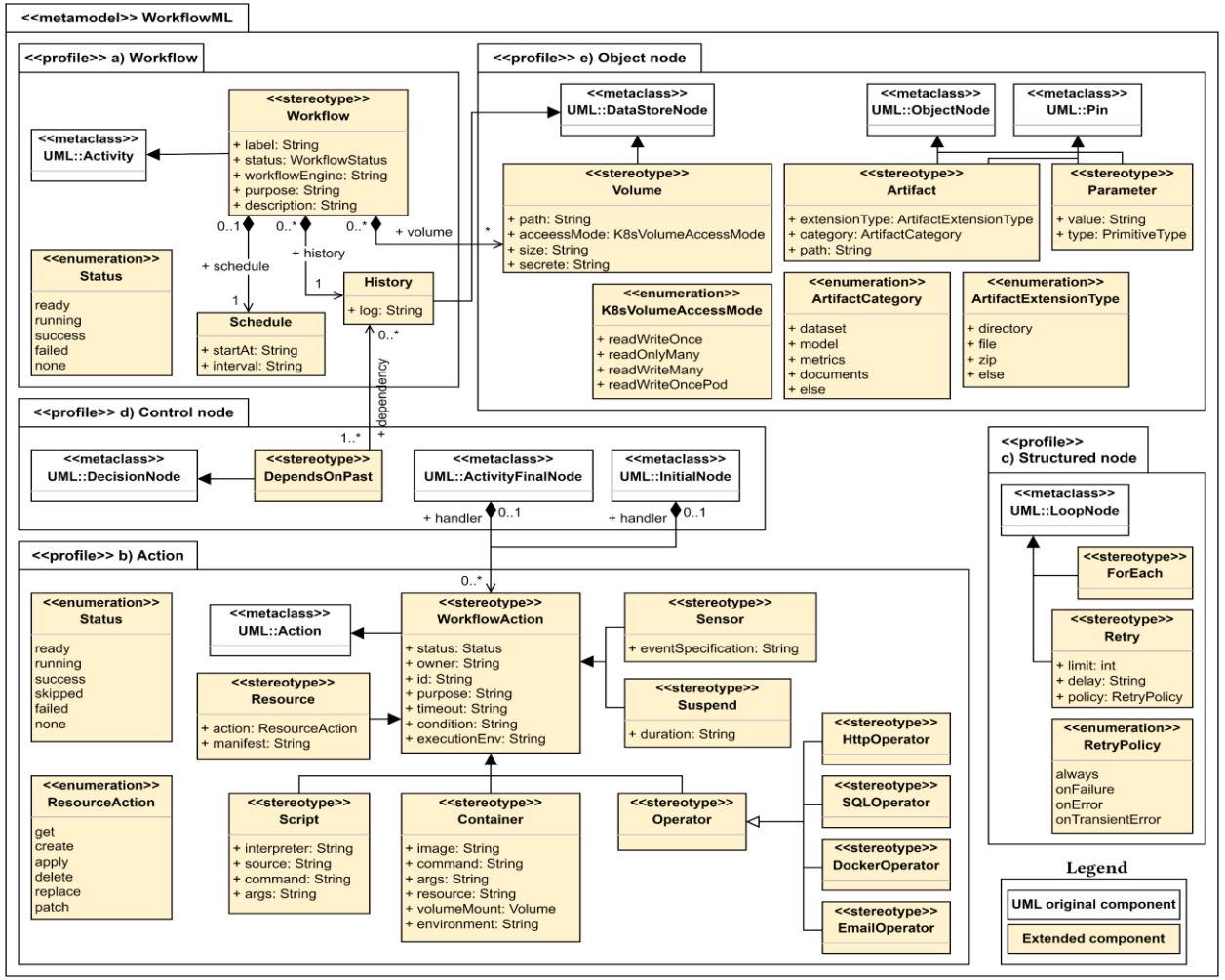
Platform-Independent Ontology Building (2/2)

- A snippet of integrated ontologies of the workflow platforms and the UML



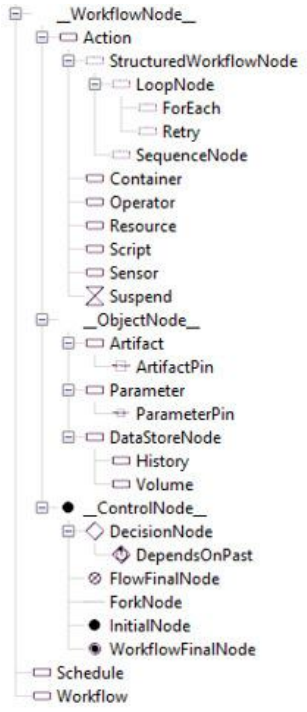
WorkflowML Metamodeling

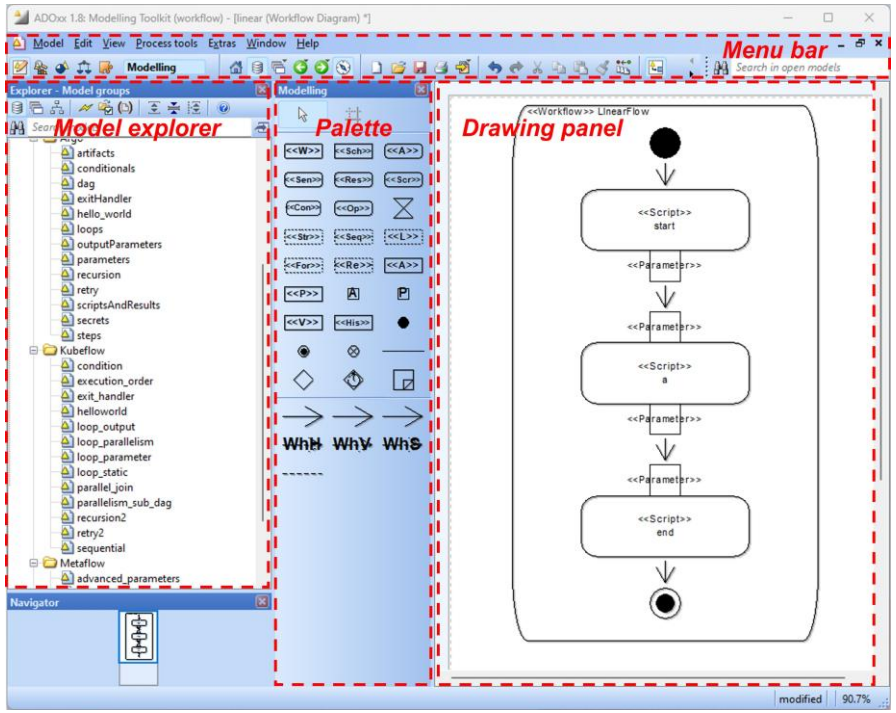
- Extend the metamodel of UML activity diagram based on the integrated workflow ontology
  - Defining stereotypes
  - Connecting the stereotypes to metaclasses
  - Defining attributes and enumerations

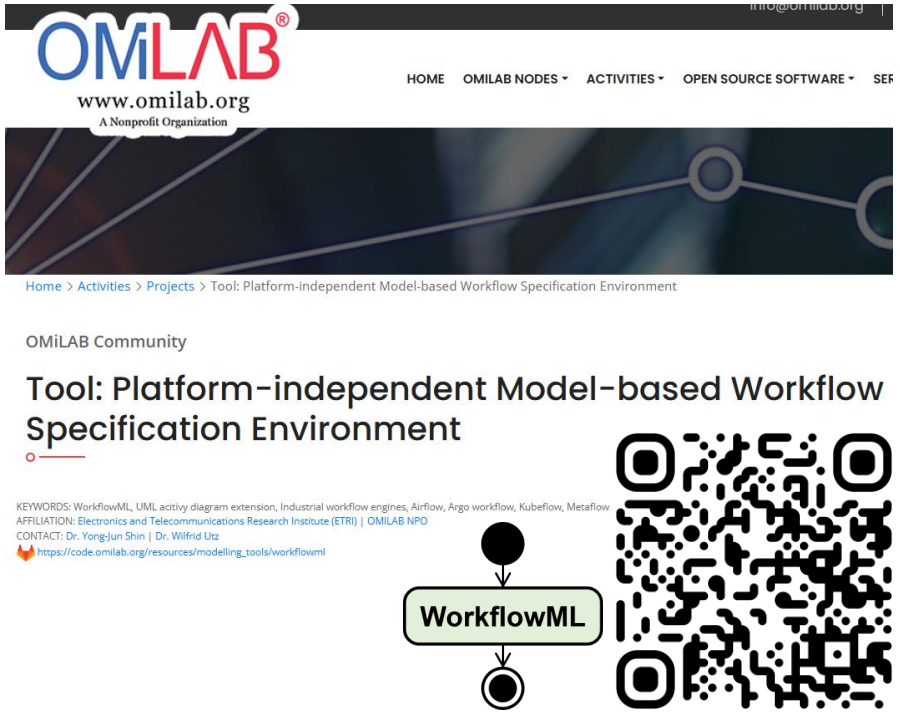


WorkflowML (Tool) Development On ADOxx

- Develop WorkflowML and its tool on ADOxx with graphical notations
  - The tool and source are available at https://www.omilab.org/workflowml/







Evaluation

- Expressiveness of the WorkflowML
- Real case studies

Software-Intensive Workflow Concept Representation

- WorkflowML improves expressiveness of UML activity diagram for software-intensive workflow specification.
  - 22 new components for workflow

WorkflowML component	#
Reused (UML activity components)	33
Extended (Domain-specific components)	22
Total	55

Table 2: Summary of the number of WorkflowML components

Domain-specific concept	UML	BPML	YAWL	WorkflowML
1. Workflow execution schedule	X	△ (Timer)	△ (Time task)	O (Schedule)
2. Workflow execution history	X	X	X	O (History)
3. Conditional execution of actions depending on the past execution	X	X	X	O (DependsOnPast)
4. Workflow data directory	△ (Central buffer)	△ (Data store)	X	O (Volulme)
5. Parallel task execution for iterable items	△ (Loop)	O (Multiple instance task)	O (Multiple instance task)	O (ForEach)
6. Retry of failed workflow tasks	△ (Loop)	△ (Loop)	△ (Loop)	O (Retry)
7. Domain-specific types of workflow tasks	△ (Action, etc.)	△ (Send, Recieve, Manual, etc.)	△ (Task, etc.)	O (Container, Script, etc.)

Table 3: Domain-specific concept expressiveness of WorkflowML and domain-general process modeling languages. O/△/X indicates that the concept is explicitly, indirectly, or not expressible, respectively, by the language.

Expressiveness Evaluation of the WorkflowML

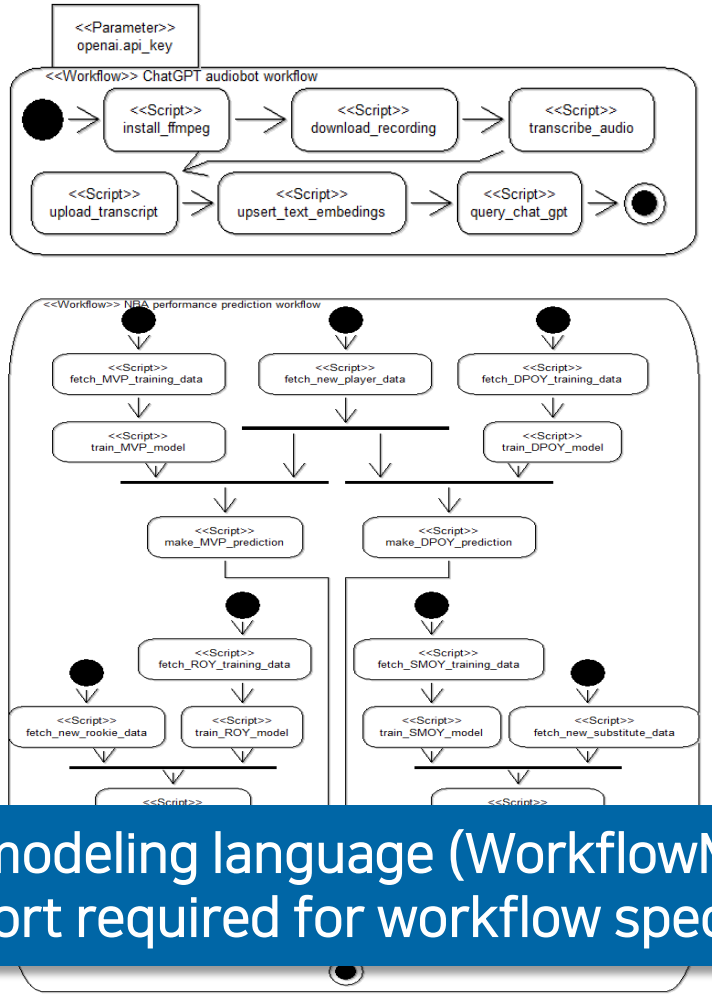
- Visual programming coverage (%)
  - The ratio of visual-programmable workflow specifications to platform-specific code-based specifications
- WorkflowML achieved an average VP coverage of 90%
  - across 42 example workflows from the four target workflow platforms

Workflow platform	Example workflow name	LoC of workflow Total (a)	LoC of workflow Modeled (b)	Visual programming coverage (b/a)(%)	Average
Airflow	Example_nested_branch_dag	56	47	83.93%	87.65%
	Example_task_group	64	55	85.94%	
	Example_bash_operator	76	66	86.84%	
	Tutorial_taskflow_api_virtualenv	87	74	85.06%	
	Example_branch_datetime_operator	104	95	91.35%	
	Example_sensors	123	106	86.18%	
	Tutorial_taskflow_api	107	94	87.85%	
	Tutorial	125	107	85.60%	
	Tutorial_dag	135	118	87.41%	
	Example_complex	220	212	96.36%	
Argo workflow	Hello_world	16	16	100.00%	96.20%
	Retrying_failed_or_errored_steps	23	19	82.61%	
	Parameters	26	26	100.00%	
	Recursion	34	34	100.00%	
	Secrets	34	34	100.00%	
	Scripts_and_results	50	34	68.00%	
	DAG	36	36	100.00%	
	Output_parameters	38	38	100.00%	
	Steps	41	41	100.00%	
	Artifacts	43	43	100.00%	
	Exit_handler	44	44	100.00%	
	Loops	49	49	100.00%	
	Conditionals	64	64	100.00%	
Kubeflow pipeline	Loop_static	26	24	92.31%	89.41%
	Loop_parameter	27	25	92.59%	
	Hello_world	30	26	86.67%	
	Loop_parallelism	30	27	90.00%	
	Loop_output	30	28	93.33%	
	Execution_order	38	34	89.47%	
	Retry	39	35	89.74%	
	Condition	46	41	89.13%	
	Exit_handler	50	46	92.00%	
	Parallel_join	53	50	94.34%	
Metaflow	Parameters	18	15	83.33%	90.48%
	Linear	20	17	85.00%	
	Advanced_parameters	23	20	86.96%	
	Foreach	27	24	88.89%	
	Branch	31	28	90.32%	
	Parallelism_sub_dag	34	30	88.24%	
	Sequential	48	45	93.75%	
	Data_flow	60	57	95.00%	
	Recursion	69	66	95.65%	

Table 4: Expressiveness evaluation results of the workflow specification of WorkflowML

Real Case Applications of the WorkflowML

- ChatGPT audiobot workflow [1]
  - VP coverage: **92.77%**
    - Total LoC: 249 lines
- NBA performance prediction workflow [2]
  - VP coverage: **95.2%**
    - Total LoC: 459 lines



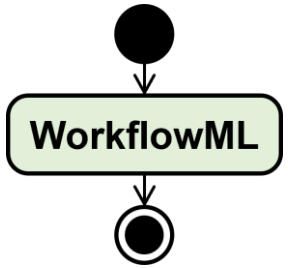
Our platform-independent workflow modeling language (WorkflowML) significantly reduces the complexity and effort required for workflow specification.

[1] https://github.com/anujkumar98/Meeting-Intelligence-Application/blob/main/Airflow/dag.py
[2] https://github.com/Sapphirine/202212-19-NBA-Player-Awards-and-Team-Performance-Prediction/blob/main/Player_DAG.py

Conclusion

Contributions

- A reusable bottom-up method for developing domain-specific modeling language using ontology integration
  - Ontology building, ontology integration, and metamodeling algorithms
- Platform-independent WoflowML based on ontologies
  - Ontology building data
  - Metamodels
- An open source WorkflowML tool
  - WorkflowML and its graphical notation implementation sources
  - 42 simple example workflow models
  - 2 real-case workflow models



ETRI OMLAB®

Thank you

A Platform-Independent Software-Intensive Workflow Modeling Language And An Open-Source Visual Programming Tool

A Bottom-Up Approach Using Ontology Integration Of Industrial Workflow Engines

Dr. Yong-Jun Shin, yjshin@etri.re.kr
Dr. Wilfrid Utz, wilfrid.utz@omilab.org

April 3, 2025