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

Yong-Jun Shin*
ETRI
Daejeon, South Korea
yjshin@etri.re.kr

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

Abstract

Many contemporary software-intensive services are developed as workflows of collaborative and interdependent tasks. Industrial workflow platforms (i.e., engines) such as Airflow and Kubeflow automatically execute and monitor the workflow specified in platform-specific code. The code-based workflow specification becomes complex and error-prone as services grow in complexity. Furthermore, differences in platform-specific workflow specifications cause inefficiencies when porting workflows between platforms, even if the different platforms handle semantically the same workflow.

In this paper, we propose a bottom-up approach for developing a platform-independent software-intensive workflow modeling language. The approach systematically extends the UML activity diagram by building platform-independent ontologies of the workflow specification from the given target industrial workflow engines. Based on the approach, we develop a platform-independent Workflow Modeling Language (WorkflowML) that covers four famous workflow engines (Airflow, Kubeflow, Argo workflow, and Metaflow). Furthermore, we implement an open-source visual programming tool for WorkflowML using the ADOxx metamodeling platform. We validate our approach by evaluating the expressiveness of WorkflowML based on modeling case studies of 42 simple workflows and two real-case workflow-based services. The evaluation results validate that WorkflowML serves as an effective common visual language for target workflow engines, supported by an open-source visual programming tool.

CCS Concepts

• **Software and its engineering → System modeling languages**.

Keywords

Workflow, Domain-specific modeling language, Metamodeling, Visual programming, Tool, ADOxx, Ontology

ACM Reference Format:
Yong-Jun Shin and Wilfrid Utz. 2025. 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. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707840

*Corresponding author

SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707840

1 Introduction

As the complexity of modern services, such as AI-enabled services, increases, workflows have been considered as an effective tool to decompose and manage tasks that a service shall accomplish [21]. Consequently, the software industry has proposed various workflow engines. This paper considers four open-source workflow engines developed by industry-leading institutions with a considerable number of users and documentation: Airflow, Kubeflow pipeline, Argo workflow, and Metaflow[1]. They commonly provide deployment, execution, orchestration, observation, and versioning of workflows. While these features are powerful, skilled developers are required to create platform-specific workflow specification codes, which are often longer than hundreds of lines, and thus are prone to errors. Additionally, in the event of business contract changes or other reasons that require the migration of developed workflows to different platforms, the transformation of the workflow specifications incurs significant costs, even if all constituent tasks are executable on two or more platforms. Although there are commercial tools that support visual programming of workflow, such as Kissflow, Camunda, and Lucidchart, they are limited by pre-built tasks that constitute a workflow, so primarily target non-expert service developers.

To overcome the challenges, we develop a modeling language that can be shared among platforms that exclusively support code-based workflow specifications for experts, so that engineers can facilitate the development of software-intensive workflows. Specifically, replacing code with models can reduce the cost of workflow specifications. Furthermore, a platform-independent modeling language allows workflows to run across various workflow engines by serving as the foundation language for platform-specific code generation.

In this context, we propose and conduct a bottom-up approach to develop a platform-independent modeling language to specify a software-intensive workflow. We build platform-independent ontology by integrating platform-specific ontologies of the target workflow engines. In this bottom-up manner, we pursue platform independence by both uniting common concepts among platforms and embracing platform-specific concepts. We extend the

[1]https://airflow.apache.org/, https://www.kubeflow.org/, https://argoproj.github.io/argo-workflows/, and https://metaflow.org/, respectively

UML activity diagram metamodel based on the integrated platform-independent ontology to develop a platform-independent Workflow Modeling Language (WorkflowML). We also develop an open-source WorkflowML visual programming tool for practitioners, using the ADOxx metamodeling platform [13].

In summary, the key contributions of this paper are as follows:

- Proposal of a bottom-up UML extension method using ontology integration for developing platform-independent modeling languages.
- Development of a platform-independent workflow modeling language (WorkflowML).
- Implementation and release of an open-source WorkflowML visual programming tool.

Furthermore, we evaluate WorkflowML by modeling 42 simple workflows and two real services in our visual programming tool.

This paper is organized as follows: Section 2 introduces related languages and tools for workflow specification. Section 3 develops the WorkflowML and the tool, and section 4 evaluates their effectiveness. Section 5 reveals threats to the validity. Finally, Section 6 concludes this paper.

2 Related Work

Workflow modeling can be achieved using general-purpose process modeling languages. For example, the UML activity diagram visually represents processes, such as workflows, illustrating the flow of actions and objects [7]. Business Process Modeling Language (BPML) enhances the clarity and efficiency of business process specifications with many detailed notations tailored for various purposes [15]. Yet Another Workflow Language (YAWL) is another modeling language based on the formal specification and methods of Petri-net [20]. These languages are not designed primarily only for software specification; instead, they serve as highly abstracted process specifications for various purposes. Consequently, directly applying these languages to software-intensive workflow engines of our interest is insufficient to develop executable services.

Some proprietary workflow modeling platforms offer visual programming of workflows through pre-built tasks in this context. For example, Kissflow[2], identified as a low code platform, supports the automation of business workflows in domains such as purchasing, human resource management or asset management. Similarly, Camunda[3] serves as a workflow automation tool for modeling, executing, and monitoring workflows built on BPML. These tools provide prebuilt task implementations (e.g., email, database operations, Google APIs, etc.), simplifying workflow execution by requiring users only to input parameters. These tools predominantly cater to non-expert service developers, potentially limiting their capabilities in developing intricate services like complicated machine learning pipelines. Contrastly, this paper aims to propose a common modeling language for platforms that exclusively supports code-based workflow specifications for experts.

In this paper, we develop the platform-independent workflow modeling language by extending the UML activity diagram. We chose UML as the foundation of our language not only because of its extension methods [7] but also the inherent complementarity among various modeling viewpoints (e.g., structures and behaviors) within UML. Some studies are similar in purpose to this paper, where activity diagrams have been used or extended for workflow modeling. Dumas and Ter Hofstede evaluated the UML activity diagram as a workflow specification language and revealed some limitations in expressiveness [6]. To improve expressiveness, Sun et al. extend it based on spatio-temporal topology that fuses control and data flow [18]. Some studies extend it for specific domains, such as the production system [3], health information system [17], and the bioinformatics system [8]. Butt et al. extended the UML activity diagram to create a workflow specification for a custom workflow engine, closely similar to this paper [4]. The purpose of extending UML in this paper, distinct from the related work, is to create a visual language that popular industrial workflow engines (specifically Airflow, Kubeflow, Argo Workflow, and Metaflow) can commonly utilize for visual workflow specification.

[2]https://kissflow.com/
[3]https://camunda.com/

3 Workflow Modeling Language And Tool Development

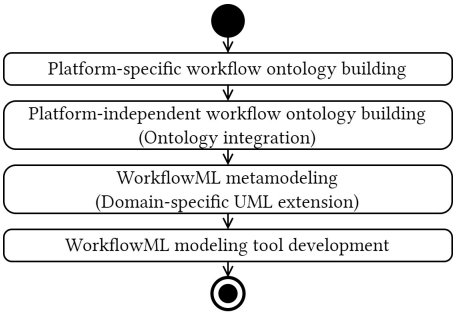

Figure 1: Platform-independent WorkflowML and tool development method

In this section, we propose and conduct a bottom-up approach for developing a platform-independent workflow modeling language. We develop a domain-specific modeling language (DSML) [1, 11] extending the UML, named Workflow Modeling Language (WorkflowML), that allows for effective workflow specification and an open-source tool that supports WorkflowML. Figure 1 shows the overall development method. Specifically, we perform four steps: 1) to collect the domain concepts thoroughly, we select some widely-known industrial workflow platforms and extract their ontologies, 2) we integrate the ontologies to embrace the workflow platforms, 3) we define a metamodel of WorkflowML extending UML activity diagram, and 4) we finally develop a modeling (i.e., visual programming) tool by implementing the WorkflowML metamodel on the ADOxx metamodeling platform. The following subsections describe each step in detail.

3.1 Platform-Specific Ontology Building

To capture the concepts of the workflow recognized by the industry, we first collect ontologies from independent workflow platforms (i.e., engines), introduced in Section 2, in a bottom-up manner. We define the ontology, propose an ontology-building process, and perform the process for each workflow platform under consideration.

The ontology is a structured corpus (e.g., graph) of a specific domain [2, 14, 22]. It systematically represents domain concepts and their relationships, so it often serves as the foundation of DSMLs [10, 12]. In this paper, we define the ontology $O = (V, E)$ as a graph, composing a pair of a set of vertices V and a set of labeled edges $E = \{(v_x, r, v_y) | v_x, v_y \in V, r \text{ is a relation label}\}$. A vertex $v \in V$ is a noun expressing a domain concept regarded as a node of an ontology graph (e.g., a *workflow* and a *task*). An edge (v_x, r, v_y) is a directed edge from a source vertex v_x to a destination vertex v_y with a relation label l. The label l is a verbal relation between v_x and v_y from the point of view of v_x (e.g. the *composition* of a *task* in a *workflow*). In addition, we adopt a lightweight ontology (LWO) [9], a specific type of ontology, whose every relationship label l is one of four relationships used in UML metamodels ((1) attribute-to-class, (2) subclassing, (3) equivalent-to, and (4) composition (relation-to)). LWO serves our purpose of using the ontology to extend the UML metamodel providing explicit alignment between the ontology and the DSML metamodel in Section 3.3.

Algorithm 1: Ontology-building process for UML extension

```
   Input  : Set of domain documents D
   Output : Ontology graph O
1  A set of vertices V ← ∅
2  A set of edges E ← ∅
3  foreach d ∈ D do
4      A set of clauses C ← ClauseExtraction(d)
5      foreach c ∈ C do
6          (s, p, o) ← SPOExtraction(c)
7          if isLWORelationship(p) then
8              V ← V + {s, o}
9              E ← E + {(s, p, o)}
10         end
11     end
12 end
13 O ← (V, E)
14 O ← conceptClustering(O)
15 return O
```

Ontology building is often expert-centric and biased, so we propose an algorithmic ontology-building process in Algorithm 1. We extend the ontology-building process of Omoronyia et al. [14] for creating LWO for UML extension. Specifically, the process takes a set of domain documents D which elaborate key concepts of the domain such as technical reports, specifications, and guidelines; it returns an ontology graph O based on the above definition. The process starts by initializing two sets: V for the vertices and E for edges, both empty (lines 1-2). It extracts all declarative clauses describing concepts and relationships of the domain from all available documents and decomposes the clauses into triples of the subject-predicate-object (SPO) to formalize concepts and relationships (lines 3-6). Then only triples whose predicate is one of the four types of LWO relationship are selected to be pushed into the output ontology (line 7). The selected triples construct the ontology graph (lines 8-13). These iterative steps collect concepts describing the workflow, represented as vertices, that fit the LWO syntax as much as possible. However, the ontology graph at this point could have inconsistent or redundant concepts and relationships due to the ambiguity and incompleteness of the given documents. Therefore, the ontology graph is refined by concept clustering [14] to improve consistency and completeness of the ontology (line 14). Specifically, we (1) merge similar concepts and relationships, (2) add or remove vertices and edges for a better understanding of the domain, and (3) connect the separated graphs by adding glue concepts. Finally, an ontology graph is returned (line 15).

Table 1: Documents used for ontology building and the number of extracted vertices of workflow engines

Workflow engine	Documents (including all subpages)	# of vetices
Airflow	https://airflow.apache.org/docs/apache-airflow/stable/ - tutorial - core-concepts	46
Argo workflow	https://argoproj.github.io/argo-workflows/ - workflow-concepts - walk-through	63
Kubeflow pipeline	https://www.kubeflow.org/docs/components/pipelines/v1/ - introduction - concepts	33
Metaflow	https://docs.metaflow.org/metaflow/basics	22

In this paper, we take four famous industrial workflow engines introduced in Section 2 (Airflow, Argo workflow, Kubeflow pipeline, and Metaflow) as sources of the workflow ontology building process. Engines support the effective execution and management of workflows for various purposes, such as machine learning or data analysis. Since the engines are already leading the industry and show considerable consensus on the concepts of the workflow, we extract the ontology from their official documents in a bottom-up and industry-driven manner. We first collect official documents (e.g., user guides and tutorials) of the workflow engines, that describe the concepts of the workflow, listed in Table 1. We then perform Algorithm 1 to generate workflow ontology graphs for each engine. The number of extracted vertices (i.e., concepts) for each engine's ontology graph is listed in Table 1. The four ontology graphs are integrated in the next subsection to build a platform-independent domain ontology, and due to lack of space, the ontology of each engine is not further described in this paper[4].

3.2 Platform-Independent Ontology Building By Ontology Integration

The platform-specific ontologies show slight differences in the concepts of the workflow, such as different types of workflow task implementations (e.g., python scripts, docker containers). Therefore, we create one common domain ontology by integrating the platform-specific ontologies to create a modeling language that can be shared by different workflow engines. We first convert the metamodel of the UML activity diagram[5] into an LWO graph. Since the relationships expressed in the UML metamodel are the same

[4]The WorkflowML tool repository: https://www.omilab.org/workflowml/
[5]https://www.omg.org/spec/UML/

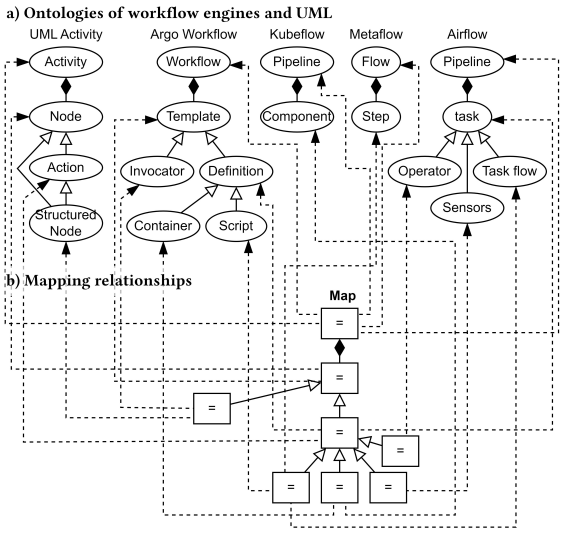

Figure 2: Integration of platform-specific ontologies

as those of LWO, the ontology of the UML activity diagram can be easily obtained by converting the metamodel into a graph form. We then integrate it as a glue ontology with the platform-specific ontologies. The UML ontology works as a skeleton during the integration and makes the integrated ontology directly mapped to the workflowML metamodel by abstracting concepts of the workflow.

We use the Vanilla ontology merging algorithm [16]. We search nodes that represent semantically the same concept redundantly defined in multiple ontologies and merge them. To do that we carefully perform the semantic mapping of the platform-specific ontologies after reading all source documents of the ontologies. Figure 2 visualizes the ontology integration. We manually combine nodes meaning the same concept into one node. For example, the 'activity', 'workflow', 'pipeline', and 'flow', which denote sequences of interdependent tasks or actions aimed at achieving a goal, are merged. Through this process, an integrated ontology graph that encompasses all concepts of different workflow engines is generated. For example, in Figure 2, Kubeflow defines a pipeline as the flow of '(containerized) components', but Argo workflow regards both 'container' and 'script' (e.g., Bash or Python) components as the task definitions of the workflow. The different components of the workflow are accepted in the integrated ontology. It is noteworthy that ontology integration does not simply increase the size of the ontology but rather merges only the common concepts, allowing the platform-specific differences to remain in the ontology. Refer to the merging algorithms we perform for further detail [16].

Figure 3 shows a snippet of the integrated ontology. The sources of the white nodes are the UML, and the sources of the yellow nodes are the workflow engines. We can see that the ontology of UML activity, which is purpose-general, has been extended. Specifically, (a) workflow, (b) action, (c) structured node, (d) control node, and (e) object node are extended by adding new subclasses or attributes. This integrated ontology is the foundation of the WorkflowML metamodel. Therefore, (a)-(e) are explicitly represented in the workflowML metamodel, each of which is specifically described in the next subsection.

3.3 WorkflowML Metamodeling

This section defines workflowML by extending the UML activity diagram metamodel[5]. UML is a purpose/domain-general modeling language but supports extensions for various purposes and domains by adding purpose/domain-specific stereotypes and attributes [7]. In addition, the lightweight ontology (LWO) which contains selected relationships based on the UML metamodel (Section 3.1) and the UML activity ontology used as the glue for ontology integration (Section 3.2) effectively support the extension of the UML metamodel. Specifically, we repeat three major steps until all concepts in the integrated domain ontology are represented in the metamodel; (1) defining workflow domain-specific stereotypes of UML activity metaclasses searching all 'subclassing' relationships in the ontology, (2) connecting all metaclasses or stereotypes searching all 'composition' relationships in the ontology, and finally (3) adding domain-specific attributes of metaclasses or stereotypes searching all 'attribute-to-class' relationships in the ontology. Additionally, discussions of authors and minor touches on the WorkflowML metamodel are performed while reviewing the metamodel.

Figure 4 shows the workflowML metamodel. UML activity diagram is extended in (a) workflow, (b) action, (c) structured node, (d) control node, and (e) object node as already mentioned in Section 3.2. Each part of Figure 4 is explained to show how WorkflowML specifies a workflow-based service.

3.3.1 Workflow. A *Workflow* is a specification of the execution flow of various actions and is also a living entity. Therefore, workflows represent their current state (e.g., ready, running). Additionally, a *Schedule* of the workflow execution can be given with the start time and execution interval of the workflow. The workflow can possess a storage *Volume* for resource sharing within the workflow and execution *History* logs. Furthermore, attributes such as labels, descriptions, and purposes are specified for workflow management.

3.3.2 Action. We define the smallest executable task that constitutes a workflow *Action*. Each action has attributes such as state, owner, ID, execution environment, execution conditions, and timeout, which is the maximum execution time limit. Concrete actions can be implemented by selecting from six types, based on the workflow engines: •*Script* for executing specific source code, •*Container* for executing containerized programs, •*Resource* for managing computational resources utilized in the workflow, •*Sensor* for waiting for specific external events, •*Suspend* for pausing execution for a certain period, and finally, •*Operator* for various domain-specific reusable action templates such as HTTP and SQL operations. The attributes of each type of action are specified in Figure 4. Additionally, WorkflowML is also an extensible language like UML, so it allows defining new action types.

3.3.3 Structured node. A set of actions can compose a structured node. Specifically, WorkflowML proposes new stereotypes of LoopNode, namely *ForEach* and *Retry*. *ForEach* comprises actions that will be iteratively executed in parallel for iterable input objects. *Retry* comprises actions that are intended to be retried a certain number

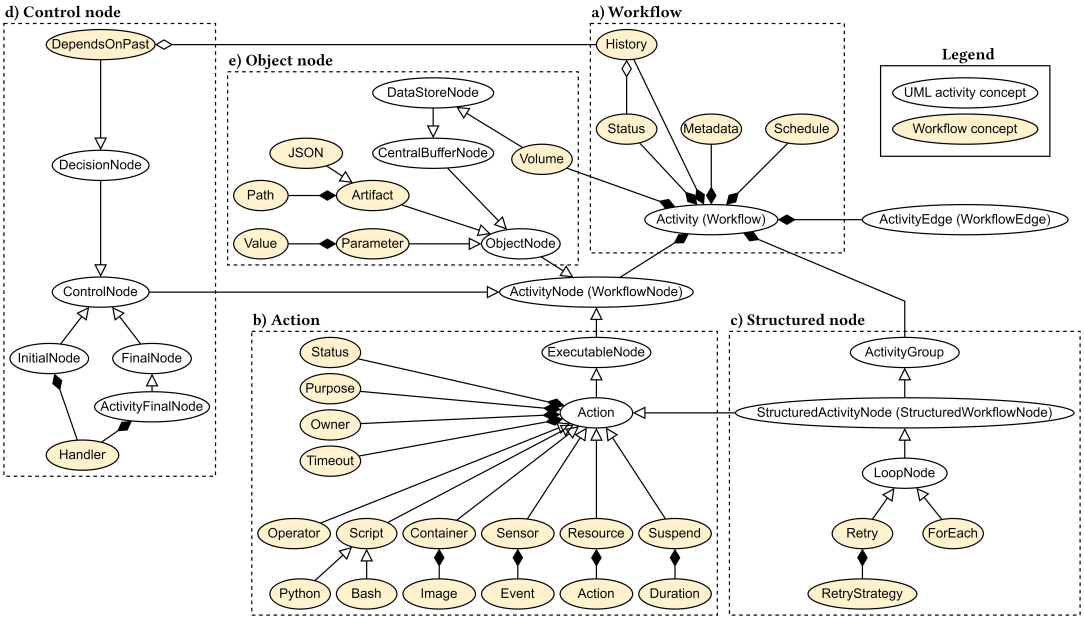

Figure 3: A snippet of integrated ontologies of the workflow platforms and the UML

of times in specific situations, such as execution failures, based on given retry policies.

3.3.4 Control node. WorkflowML not only reuses control nodes of UML (e.g., decision and fork nodes) but also proposes an extension of the decision node stereotype *DependsOnPast*. It represents branching based on the past execution results of the workflow, so it refers to the history of the workflow. Additionally, WorkflowML allows the initial and final nodes of a workflow to have specific actions as handlers.

3.3.5 Object node. Actions in the Workflow interact with input and output objects, which can be broadly categorized into *Artifact*s and *Parameter*s. Parameters pass simple values, while artifacts pass objects that cannot be conveyed as simple parameters but are stored as structured data at specific paths. Artifacts can be further specified with their extensions and categories. Additionally, the storage in which the actions of the workflow can share data is defined as a *Volume*, which is a new stereotype of DataStoreNode.

The detailed attributes and enumerations of the extended components in WorkflowML are listed in Figure 4.

3.4 Modeling Tool Development

We develop an open-source modeling tool supporting our WorkflowML using ADOxx metamodeling platform developed by OMiLab NPO [13]. ADOxx supports implementing graphical notations and grammar of the DSML. We defined metamodel and graphical notations of WorkflowML in ADOxx by inheriting UML metaclass library. The implementation result can be downloaded as an ADOxx library file from our repository[4].

Users can use or extend our WorkflowML modeling tool by loading the WorkflowML library file on ADOxx, an open-use platform. Figure 5 illustrates the WorkflowML modeling tool, comprising a menu bar, a model explorer, a palette of WorkflowML components, and a drawing panel. By selecting WorkflowML components from the palette, users can specify a concrete workflow on the drawing panel. Users can also edit the attributes of the components in detail by double-clicking on them. The model can be extracted not only in image form (JPG, etc.) but also in a machine-readable format (XML) for further processing, such as automatic workflow code generation. To aid users in becoming familiar with the WorkflowML modeling tool, we provide example models of 42 simple workflows and two real-case workflows, along with the WorkflowML library[4]. These examples are used in the following evaluation section.

4 Evaluation

We have developed a WorkflowML and its accompanying tool. In this section, we aim to evaluate whether WorkflowML is an effective DSML that promotes model-based workflow specification. Specifically, we answer the following three research questions (RQ):

- RQ1: Does WorkflowML reflect concepts of software-intensive workflow well?
- RQ2: Does WorkflowML support platform-independent model-based workflow specification?
- RQ3: Can WorkflowML be applied in the specification of real-world workflow-based services?

The following subsections are dedicated to each research question.

4.1 Representation of Software-Intensive Workflow Concepts

WorkflowML is a modeling language to specify software-intensive workflows extended from UML activity diagram. Table 2 summarizes the number of WorkflowML components (metaclasses and stereotypes) categorized into those reused from UML and those

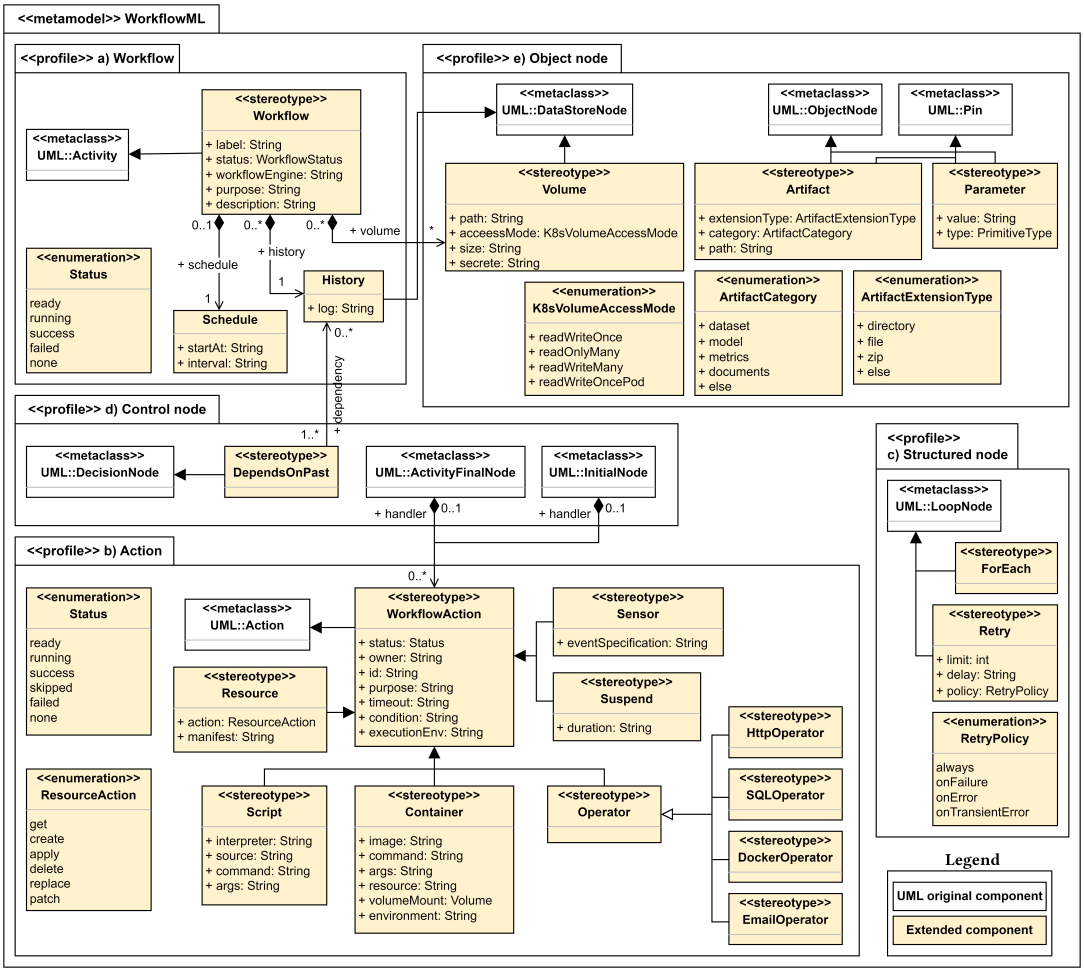

Figure 4: WorkflowML metamodel

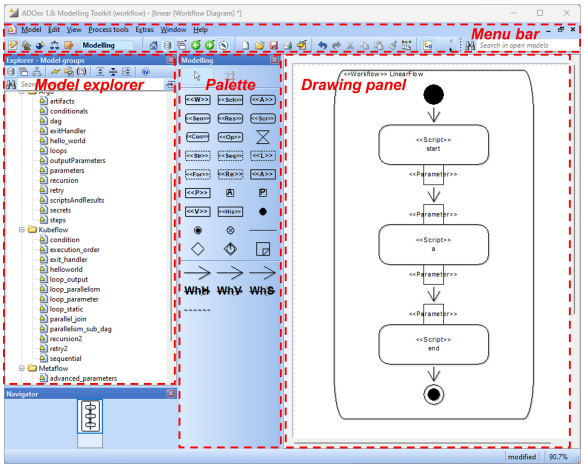

Figure 5: The open-source WorkflowML modeling tool

Table 2: Summary of the number of WorkflowML components

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

extended for workflow domain purposes. We selected 33 core metaclasses of UML activity and reused them in WorkflowML. Then we developed 22 new components extended from the UML components for software-intensive workflow specifications.

In this subsection, we qualitatively assess whether WorkflowML reflects domain-specific (i.e., software-intensive workflow) concepts well. First, we identify and enumerate the concepts commonly addressed in the software-intensive workflow specifications of the industrial workflow engines covered in this paper in Table 3. We intentionally refrain from listing concepts typically addressed in general process modeling, such as a specification of task flows or hierarchical tasks, in the table. Instead, we exclusively enumerate

concepts specialized for modern software-intensive workflow engines, which are selected under the authors' discussion based on the documents of workflow engines. We then assess how WorkflowML concretely represents these concepts. For comparison, we also assess the expressiveness of well-known domain-general process modeling languages. Specifically, we compare WorkflowML with UML activity diagram[5], Business Process Model and Language (BPML) [5], and Yet Another Workflow Language (YAWL) [19] according to their official specifications. Table 3 presents the evaluation results, where each row represents the evaluation result of WorkflowML and the other languages in terms of the coverage of concepts in the workflow specification. Concepts explicitly specified by corresponding modeling components (i.e., metaclasses and stereotypes) are marked with 'O', concepts indirectly represented by sets of abstract modeling components are marked with '△', and concepts not covered by modeling languages are marked with 'X'. Additionally, modeling components used to model the concepts are noted.

Table 3 illustrates that WorkflowML is a domain-specific modeling language defined to model the concepts covered by industrial workflow engines as concrete modeling components, compared to the other domain-general process modeling languages. The evaluation results for each concept in the modeling languages are as follows: 1) The workflow engines support the automatic execution of workflows iteratively according to predefined schedules. Therefore, while BPML and YAWL allow independent tasks to have time-based triggers, WorkflowML enables a workflow (i.e., a flow of tasks) itself to explicitly possess schedule nodes. 2-3) The engines store the execution histories of workflows. In addition, workflow executions may depend on the past execution results (i.e., histories). For instance, tasks successfully executed yesterday might not be rerun today. UML, BPML, and YAWL do not address these concepts, but WorkflowML explicitly defines a history repository node for storing past execution results and a conditional node that depends on the history. 4) Constituent tasks of a workflow managed by the engines can have a shared data directory. While UML and BPML can abstractly model such shared repositories using 'Central buffer' or 'Data store' nodes, WorkflowML concretely models the shared directory as the 'Volume' node defining concrete attributes including storage size, path, and access mode. 5) Workflow engines can also specify the repetitive execution of tasks for iterable items. 'Multiple instance task' in BPML and YAWL models such repetitive executions, and WorkflowML also specifies this as the 'ForEach' node. 6) Workflow engines provide retry strategies for failed task executions. Modelers can indirectly express such retry strategies using loops of UML, BPML, and YAWL. On the other hand, WorkflowML concretely specifies these retry strategies as 'Retry' nodes. 7) Finally, workflow engines support the implementation of software-intensive tasks. UML, BPML, and YAWL support abstract task specifications, but WorkflowML provides various types of software-intensive tasks supported by industrial workflow engines, such as containers and scripts defined in Section 3.3. Therefore, users can easily develop executable tasks by filling in the attributes of these WorkflowML components. In this manner, WorkflowML is defined to explicitly specify software-intensive workflows for industrial workflow engines.

4.2 Effectiveness of Platform-Independent Model-Based Workflow Specification

Previously, we qualitatively evaluated that WorkflowML covers the concepts in software-intensive workflow specifications. Now, we assess whether WorkflowML has sufficient expressiveness to support model-based workflow development for industrial workflow platforms. The code-based workflow specifications of the platforms often suffer from low readability and are prone to errors. We believe the model-based workflow specification can greatly facilitate efficient specification, development, analysis, and management of workflow-based services. To achieve this, WorkflowML should be able to sufficiently specify as much information as possible that is required to execute the services in the workflow platforms.

To measure WorkflowML's expressiveness, we defined an "visual programming coverage" (*VP coverage*) as the ratio of lines of code (LoC) explicitly specified in the workflow model to the total LoC in the code-based workflow specification. For example, if a specific workflow service is specified by 100 LoC, and WorkflowML can specify 90 lines among the 100 lines in a model, the VP coverage of WorkflowML for the service would be calculated as 90%. To evaluate the expressiveness, we collected all core example workflows introduced in tutorials of the workflow engines, as shown in Table 4. Then, we manually model each of these workflows using WorkflowML and count the LoC that could be explicitly specified in the workflow model compared to the LoC of the original code-based specification. For instance, in Table 4, the "example_nested_branch_dag" workflow from Airflow is originally specified with 56 lines of code (LoC), out of which 47 lines can be specified in our workflow model, while nine lines cannot.

Table 4 displays the VP coverage of WorkflowML for each of the 42 example workflows we collected. In the collected cases, WorkflowML achieved an average VP coverage of 90% for expressiveness. It demonstrates that WorkflowML can specify a substantial portion of the necessary content for workflow development within the model. However, WorkflowML is a platform-independent modeling language, so it does not specify all platform-specific details. For example, WorkflowML does not cover some Kubernetes-specific attributes used in Argo workflow's "Retrying_failed_or_errored_steps" and "Scripts_and_results" examples, while it can model most of the the other examples of Argo workflow. The remainings not covered are most typical redundancies or syntactic patterns for each platform, so they can be tailored to each platform based on WorkflowML with little effort. More importantly, WorkflowML enables visual workflow specification with high VP coverages.

Simultaneously, we have confirmed that it is possible to generate workflow specifications for various workflow engines using a platform-independent modeling language, WorkflowML. As WorkflowML is based on an integrated domain ontology that merges platform-specific ontologies by semantic matching, it can sufficiently specify workflows for all engines of interest. This is shown in the high VP coverages for all workflow engines. Therefore, WorklfowML is an effective modeling language of software-intensive workflow that can be commonly used for the four workflow engines under test. In addition, our bottom-up approach is also validated as effective in systematically building a platform-independent modeling language.

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.

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.)

In summary, WorkflowML can effectively support model-based workflow development by replacing the majority of workflow specification contents with models instead of code. This suggests the potential for creating a model-based framework for developing workflow-based services using WorkflowML. Furthermore, WorkflowML can be used as a common workflow specification language for industrial workflow engines.

4.3 Real Case Applications of the WorkflowML

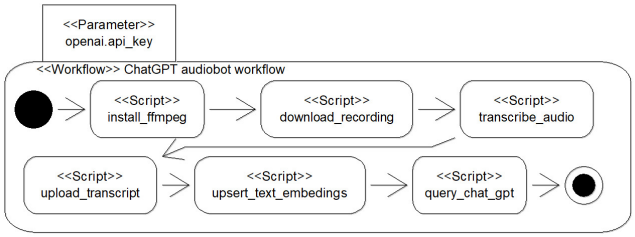

Figure 6: ChatGPT audio-bot workflow

Finally, we conduct case studies applying WorkflowML to real workflow-based services. We found two interesting workflow-based services on GitHub that were originally specified using Airflow code, and we modeled them using WorkflowML. Figure 6 and 7 show the model-based specification results for both cases.

The first case is a workflow-based ChatGPT audio bot for the meeting assistance application shown in Figure 6, which was originally specified in 249 LoC[6]. The workflow involves a series of actions, including installing an audio processing library, downloading recordings, transcribing and uploading audio files, performing text embedding, and querying ChatGPT. This entire workflow is visually specified using WorkflowML, making it easy for developers to design and manage the audio bot service effectively. As evaluated in RQ2, the VP coverage of WorkflowML for this case is 92.77%, as 231 lines out of the 249 lines in the workflow specification could be modeled.

The second case shown in Figure 7 is a machine learning workflow service for performance prediction of NBA teams and players

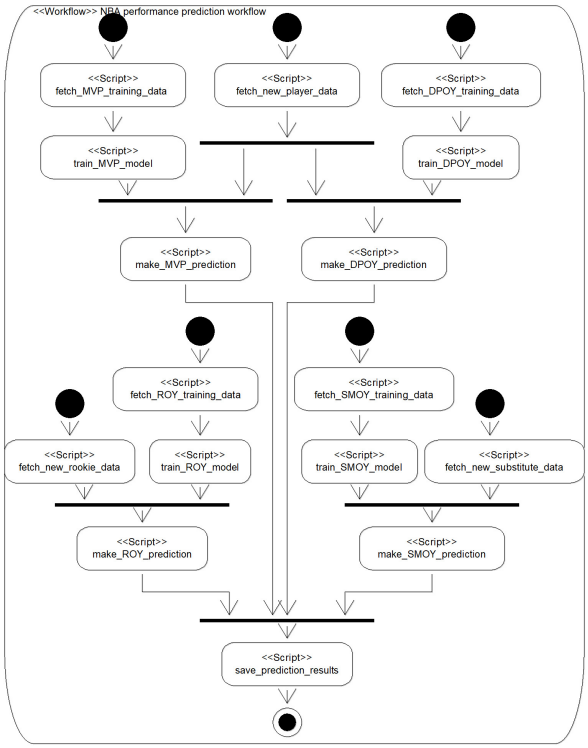

Figure 7: NBA team performance prediction workflow

based on multiple data sources in parallel, which is originally specified in 459 LoC[7]. Whenever new data about players and teams come in, the workflow is triggered to run. The performance prediction is done in parallel for Most Valuable Player (MVP), Defensive Player of the Year (DPOY), Rookie of the Year (ROY), and Sixth Man of the Year (SMOY) awards. The results from all predictions are then aggregated and stored. This service includes numerous parallel machine learning tasks, and WorkflowML could explicitly model the dependencies of the parallelism. In this case, WorkflowML achieved an VP coverage of 95.2%, as 437 lines out of 459 lines of code could be specified in the model.

[6]ChatGPT audio bot: https://github.com/anujkumar98/Meeting-Intelligence-Application/blob/main/Airflow/dag.py

[7]NBA performance prediction: https://github.com/Sapphirine/202212-19-NBA-Player-Awards-and-Team-Performance-Prediction/blob/main/Player_DAG.py

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

Workflow platform	Example workflow name	LoC of workflow		Visual programming coverage (b/a)(%)	Average
		Total (a)	Modeled (b)		
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%	

We summarize the evaluation results based on our experience of applying WorkflowML to real cases.

First, as indicated by RQ1 and RQ2, WorkflowML shows a promising level of expressiveness, supporting the development of software-intensive workflow-based services. It enables the specification of key aspects of workflows, such as actions, dependencies, and input objects for execution. These results provide some validation for our bottom-up approach to developing a workflow modeling language based on industrial workflow engines.

Second, our tool has the potential to be beneficial for specifying software-intensive workflows, which are often implemented as code in popular workflow engines. For example, the NBA team performance prediction workflow we modeled, originally written in 459 lines of code, includes 22 action dependencies. Understanding the overall structure from code alone can be challenging. Our language and tool significantly reduce this complexity by providing a visual representation of workflows. Additionally, model-based workflow specifications can serve as useful communication artifacts for various stakeholders.

Lastly, WorkflowML has capacity to serve as a common language for workflow specifications across different platforms. While code-based specifications tend to be constrained by platform-dependent grammar, WorkflowML offers a more consistent model-based specification that can be adapted for various platforms with relatively little effort. For instance, the two workflow models we developed for Airflow could be converted into workflow code for different

engines by mapping the contents of the model to the specification grammar of another target engine.

5 Threats To Validity

An internal threat to the validity of WorkflowML is that its quality relies on the underlying domain ontology. To address this, we constructed domain ontologies using various documents and well-established ontology building and merging algorithms [9, 14, 16]. We also redefined the ontology-building process as an algorithm in Section 3.1, and we have made the raw data publicly available for reproducibility[4].

Another internal threat, which is common in related works, is the subjectivity in evaluating WorkflowML. To mitigate this, we conducted both qualitative and quantitative evaluations. We compared WorkflowML against popular process modeling languages to see if it adequately represented industrial workflow concepts. We also introduced a "visual programming coverage" to quantitatively measure how well WorkflowML converts code-based workflow specifications into model-based representations. While this evaluation confirms WorkflowML's domain specificity, further usability testing covering human factors is planned for future work.

A potential external threat is that the evaluation may not generalize well due to the limited cases used. To minimize this risk, we modeled 42 simple tutorial workflows from four workflow engines (RQ2). Although these examples are basic, they can be combined to create complex real-case workflows, making the results applicable to real-world scenarios. Additionally, in RQ3, we validated WorkflowML using two real-case workflows from open-source repositories. As WorkflowML is extensible like UML, it can be tailored for specific needs, making it a versatile tool for defining software-intensive workflows in various industrial settings.

6 Conclusion

We have introduced a bottom-up approach for developing a platform-independent modeling language for software-intensive workflow specifications. With this approach, we have developed the Workflow Modeling Language (WorkflowML) by extending the UML activity diagram. Furthermore, we have released an open-source visual programming tool utilizing the ADOxx metamodeling platform. WorkflowML is built upon an integrated ontology that captures the concepts of software-intensive workflow specifications from four popular industrial workflow engines (Airflow, Argo workflow, Kubeflow, and Metaflow). Compared to well-known domain-general process modeling languages, WorkflowML explicitly represents the key concepts of software-intensive workflow specifications. The expressiveness and applicability of WorkflowML, along with its associated tool, have been validated through case studies involving 42 simple workflows and two real-case services. By replacing code-based workflow specifications in industrial workflow engines with model-based specifications, WorkflowML enhances the efficiency of workflow development and management. Moreover, it provides a language that can be shared across various platforms, effectively supporting software compatibility and migration. In our future research, we plan to utilize WorkflowML for agile machine learning operations (MLOps) in industry, such as the model-based continuous integration of autonomous driving workflows.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00406245, Development of Software-Defined Infrastructure Technologies for Future Mobility)

References

[1] Young-Min Baek, Zelalem Mihret, Yong-Jun Shin, and Doo-Hwan Bae. 2020. A modeling method for model-based analysis and design of a system-of-systems. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 336–345.

[2] Young-Min Baek, Jiyoung Song, Yong-Jun Shin, Sumin Park, and Doo-Hwan Bae. 2018. A meta-model for representing system-of-systems ontologies *(SESoS '18)*. Association for Computing Machinery, New York, NY, USA, 1–7.

[3] Ricardo Melo Bastos and Duncan Dubugras A Ruiz. 2002. Extending UML activity diagram for workflow modeling in production systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, 3786–3795.

[4] Anila Sahar Butt, Nicholas J Car, and Peter Fitch. 2020. Towards Ontology Driven Provenance in Scientific Workflow Engine.. In *MODELSWARD*. 105–115.

[5] Michele Chinosi and Alberto Trombetta. 2012. BPMN: An introduction to the standard. *Computer Standards & Interfaces* 34, 1 (2012), 124–134.

[6] Marlon Dumas and Arthur HM Ter Hofstede. 2001. UML activity diagrams as a workflow specification language. In *International conference on the unified modeling language*. Springer, 76–90.

[7] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. 2004. An introduction to UML profiles. *UML and Model Engineering* 2, 6-13 (2004), 72.

[8] Laiz Heckmann Barbalho de Figueroa and Rema Salman. 2019. A UML Activity Diagram Extension and Template for Bioinformatics Workflows: A Design Science Study. (2019).

[9] Constantin Hildebrandt, Aljosha Köcher, Christof Küstner, Carlos-Manuel López-Enríquez, Andreas W Müller, Birte Caesar, Claas Steffen Gundlach, and Alexander Fay. 2020. Ontology building for cyber–physical systems: Application in the manufacturing domain. *IEEE Transactions on Automation Science and Engineering* 17, 3 (2020), 1266–1282.

[10] Dimitris Karagiannis. 2015. Agile modeling method engineering. In *Proceedings of the 19th panhellenic conference on informatics*. 5–10.

[11] Dimitris Karagiannis, Moonkun Lee, Knut Hinkelmann, and Wilfrid Utz. 2022. *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*. Springer Nature.

[12] Grigory Kulagin, Ivan Ermakov, and Lyudmila Lyadova. 2022. Ontology-Based Development of Domain-Specific Languages via Customizing Base Language. In *2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, 1–6.

[13] OMiLAB. 2022. Development of Conceptual Models and Realization of Modelling Tools Within the ADOxx Meta-Modelling Environment: A Living Paper. In *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*. Springer, 23–40.

[14] Inah Omoronyia, Guttorm Sindre, Tor Stålhane, Stefan Biffl, Thomas Moser, and Wikan Sunindyo. 2010. A domain ontology building process for guiding requirements elicitation. In *Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, REFSQ 2010, Essen, Germany, June 30–July 2, 2010. Proceedings 16*. Springer, 188–202.

[15] Gregor Polančič. 2020. BPMN-L: A BPMN extension for modeling of process landscapes. *Computers in Industry* 121 (2020), 103276.

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

[17] Stergiani Spyrou, Panagiotis Bamidis, Kostas Pappas, and Nikos Maglaveras. 2005. Extending UML activity diagrams for workflow modelling with clinical documents in regional health information systems. In *Connecting Medical Informatics and Bioinformatics: Proceedings of the 19th Medical Informatics Europe Conference (MIE2005). Geneva, Switzerland*. 1160–1165.

[18] Xiaoya Sun, Liang Hu, and Xilong Che. 2019. Scientific Workflow: Modeling Methods and Management System. In *Journal of Physics: Conference Series*, Vol. 1168. IOP Publishing, 032023.

[19] Arthur HM Ter Hofstede, Wil MP Van der Aalst, Michael Adams, and Nick Russell. 2009. *Modern Business Process Automation: YAWL and its support environment*. Springer Science & Business Media.

[20] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 2005. YAWL: yet another workflow language. *Information systems* 30, 4 (2005), 245–275.

[21] Wattana Viriyasitavat, Li Da Xu, Gaurav Dhiman, Assadaporn Sapsomboon, Vitara Pungpapong, and Zhuming Bi. 2021. Service workflow: State-of-the-Art and future trends. *IEEE Transactions on Services Computing* (2021).

[22] DA Wagner, M Chodas, M Elaasar, JS Jenkins, and N Rouquette. 2023. Ontological Metamodeling and Analysis Using openCAESAR. In *Handbook of Model-Based Systems Engineering*. Springer, 925–954.