Implementation-independent Knowledge Graph Construction Workflows using FnO Composition

Gertjan De Mulder[1], Ben De Meester[1]

[1]IDLab, Department of Electronics and Information Systems,
Ghent University – imec, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium

Abstract

Knowledge Graph construction is typically a task within larger workflows, with a tight coupling between the abstract workflow and its execution. Mapping languages increase interoperability and reproducibility of the mapping process, however, this should be extended over the entire Knowledge Graph construction workflow. In this paper, we introduce an interoperable and reproducible solution for defining Knowledge Graph construction workflows leveraging Semantic Web technologies. We describe how a data flow workflow can be described interoperable (i.e., independent from the underlying technology stack) and reproducible (i.e., with detailed provenance) by composing semantic abstract function descriptions; and how such a semantic workflow can be automatically executed across technology stacks. We demonstrate that composing functions using the Function Ontology allows for functional descriptions of entire workflows, automatically executable using a Function Ontology Handler implementation. The semantic descriptions allow for interoperable workflows, the alignment with P-PLAN and PROV-O allows for reproducibility, and the mapping to concrete implementations allows for automatic execution.

Keywords

Workflows, RDF construction

1. Introduction

Knowledge Graph (KG) construction – i.e., RDF graph construction – is typically a task within larger workflows. The construction of a KG itself can also be considered an overarching and more complex task that is composed of smaller tasks, e.g., extracting data from a database, mapping it to RDF, and publishing it using a web API. Such a process – i.e., a set of tasks that can be automated – can be facilitated using a workflow system.

Mapping languages increase interoperability (i.e., being independent from the processor that executes them) and reproducibility (i.e., the described transformations can be reproduced using detailed provenance data) of the mapping process. However, this is not extended to the entire KG construction workflow. Existing KG construction workflows typically integrate the mapping process in a hard-coded workflow where there is a tight coupling between abstract workflow and execution, and where the lack of detailed descriptions of tasks, and their executions, inhibits the workflow from being reproduced.

KGCW'22: International Workshop on Knowledge Graph Construction, May 30, 2022, Crete, GRE
gertjan.demulder@ugent.be (G. De Mulder); ben.demeester@ugent.be (B. De Meester)
https://ben.de-meester.org/#me (B. De Meester)
0000-0001-7445-1881 (G. De Mulder); 0000-0003-0248-0987 (B. De Meester)

Interoperability diminishes when there is a tight coupling between the abstract workflow and its execution [1]. The lack of interoperability inhibits using different tools for a task, making it harder to adapt to changing requirements and constraints. For example, Tool A might initially suffice for the KG construction task given the source data size. Later on, the data size might become unmanageable for Tool A. Tool B can handle larger data sets, however, the lack of interoperability prevents the flexibility in switching from one tool to the other.

Reproducibility diminishes when details about the workflow execution are unavailable. The results produced by a workflow rely on the applied tasks and their parameters, however, if these details are not captured, it prevents other users from exactly reproducing the workflow.

In this paper, we represent tasks within a workflow using implementation-independent semantic function descriptions, and workflows as compositions of such functions. By decoupling the abstract workflow from its concrete implementation, we provide interoperability between tasks and the tools that execute them. Users can focus on the overarching task for which the workflow was created, e.g. managing the KG construction life cycle using different mapping processors that generate RDF, and different endpoints on which the RDF is published. By being aligned with the W3C standard PROV, the semantic function descriptions allow for capturing workflow execution details in a standardized way and subsequently facilitate its reproducibility.

Section 2 presents related work. In Section 3, we first show how interoperability between tasks and tools within a workflow can be achieved through the composition of declarative function descriptions and how this allows detailed provenance capture, and then provide an implementation by leveraging the Function Ontology (FnO) [2] as a model to describe and compose tasks within a data flow workflow, decoupled from the tools that are used. We introduce a motivating use case in Section 4, and demonstrate the resulting workflow in Section 5. We conclude in Section 6 and give additional pointers for future work.

2. Related work

In this section, we discuss existing KG construction workflows, and the characteristics that increase workflow systems' interoperability and reproducibility.

Mapping languages can provide features to cover many steps within the KG construction process, i.e., not only specify how to map to RDF, but also how to extract data from different data sources [3], and how to publish using various methods [4]. Even when mapping languages provide enough features to be deemed end-to-end, executing a KG construction exists within a wider context, e.g., being part of a Knowledge Graph Lifecycle [5], or as a collection of subtasks to allow for optimization [6]. As such, even though KG construction rules can be described interoperably using, e.g., a mapping language, its position within the wider and narrower tasks makes it interpretable as being (a part of) a workflow.

The state of the art on workflow languages focuses on a variety of features, of which we focus on the combination of being a declarative approach, separating abstract tasks from concrete implementations, and capturing execution provenance information.

An imperative paradigm (i.e., exact ordered instructions defining how to execute a program) is suitable for processes that are unlikely to change, however, a **declarative approach is recommended** when workflows resemble processes with changing requirements and constraints that

require them to be executed in different ways. Declarative paradigms can be used to represent data flow, i.e., the data dependencies between tasks, and are more robust to change as they describe what needs to be done, instead of how [7].

Interoperability diminishes when there is a tight coupling between tasks and implementations [1], e.g., when using ad hoc approaches. Thus, the **separation of description and implementation** is crucial to interoperability [8].

Provenance information including the execution details is required for reproducibility [9]: unambiguous and fine-grained descriptions of the workflow tasks [10].

Several workflow specifications exist, and can be divided into two parts: *executable* specifications (i.e. the descriptions contain sufficient details to be automatically executed), such as the Common Workflow Language (CWL), and Workflow Description Language (WDL); and *descriptive* specifications such as P-PLAN and the Open Provenance Model for Workflows (OPMW). To the best of our knowledge, no specification exists that can describe the provenance workflows in sufficient details whilst also focusing on automatic executability and remaining loosely coupled from the implementation. CWL allows for describing a computational workflow and the command-line tools used for executing its tasks [11]. CWL prioritizes on making it straightforward to re-run a workflow on a different machine, requiring explicit execution definitions, resulting in a tight coupling between tasks and implementations. WDL focusses on readability, making it easier to learn, but is less expressive [12]. P-PLAN extends the W3C standard PROV [13]. It allows for describing workflow steps and link them to execution traces, and was applied in projects that focus on interoperability [14] and reproducibility [10]. OPMW is an extension of P-PLAN [14]: a simple interchange format for representing workflows at different levels of granularity (ie. abstract model, instances, executions).

The Function Ontology (FnO) [2] presents a similar approach towards interoperable data transformations using Semantic Web technologies. An implementation-independent function description allows for a decoupled architecture that separates the definition from its execution, and the inputs and outputs of a function are explicitly described. Furthermore, a recent update to FnO includes composition: compose a new function from other functions.

3. Method and Implementation

In this paper we put forward our approach towards interoperable and reproducible workflows through declarative descriptions that separate description from implementation in a standardized language. This provides the flexibility of tasks being implemented by different tools and allows detailed provenance capture.

We represent tasks using implementation-independent semantic function descriptions: being semantically described makes tasks uniquely identifiable, unambiguously defined, universally discoverable, and linkable [2]. We then represent workflows as compositions of functions. For simplicity, we currently only consider sequential execution of tasks. The data flow between composed tasks is represented by input and output mappings between functions. Such a composition mapping describes how an input or output of one function is linked to the input or output of another function. For example, within a KG construction workflow this is needed to connect the output of an RDF generation task to the input of the subsequent publishing task.

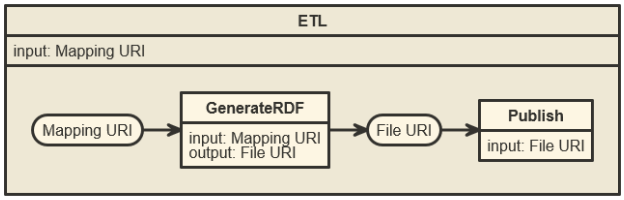

Figure 1: Workflow defined as a function composition: GenerateRDF and Publish are function descriptions of the tasks that share parameters, while ETL is a description of the constituting workflow.

An example of a function composition for an ETL workflow is illustrated in Figure 1.

We consider the Function Ontology (FnO)[1] as the model to describe functions and function compositions representing tasks and workflows, respectively. FnO allows linking abstract functions to concrete implementations, hence, providing sufficient detail to be directly executed. The recently-added specification of how to compose functions allows us to describe complete workflows using a single and simple language. We clarify using a working example of an Extract-Transform-Load (ETL) workflow comprising two tasks: i) generating RDF; and ii) publishing the generated RDF. Due to space restrictions only excerpts of the descriptions are shown.

Our workflow system requires us to create semantic descriptions of each task. First, we define the task of generating RDF as a function that takes the URI to a mapping, and the URI to which the result should be written. We make use of the RML mapping language to have an interoperable RDF generation step. Secondly, we define the publishing task as a function which takes the URI to the generated RDF data as input parameter and outputs a URI to the endpoint through which it is published. These descriptions are shown in Listing 1.

```
1  @prefix fno: <https://w3id.org/function/ontology#> .
2  @prefix fns: <http://example.com/functions#> .
3
4  fns:generateRDF a fno:Function ;
5      fno:expects ( fns:fpathMappingParameter ) ; fno:returns ( fns:fpathOutputParameter ) .
6
7  fns:publish a fno:Function ;
8      fno:expects ( fns:inputRDFParameter ) ;     fno:returns ( fns:returnOutput ) .
```

Listing 1: FnO task descriptions for generating RDF and publishing RDF

We describe an overarching ETL task as the composition of these two functions, illustrated in Listing 2, using fnoc:CompositionMapping. fnoc:Composition links the output of the first task to the second task by means of a fnoc:CompositionMapping. A fnoc:CompositionMapping links a source fnoc:CompositionMappingEndpoint to a target fnoc:CompositionMappingEndpoint by means of fnoc:mapFrom and fnoc:mapTo, respectively. A fnoc:CompositionMappingEndpoint specifies the fno:Function and either fno:Parameter or fno:Output that are being mapped. Note that, depending on how parameters and outputs are mapped, different composition mappings are possible:

- fno:Parameter to fno:Parameter. This is needed to link the parameters of the composed function to parameters of its constituent functions. In Figure 1, for example, this is needed

[1] https://w3id.org/function/spec/

to link the mapping parameter of the overarching ETL function to the mapping parameter of the GenerateRDF function.

- fno:Output to fno:Parameter. Needed to define the data flow between functions. In Figure 1, for example, this is needed to link the output of the GenerateRDF to the input of the Publish.
- fno:Output to fno:Output. This is needed to link outputs of constituent functions to outputs of the composed function. In Figure 1, for example, this is needed to link the output of Publish to the output of the overarching ETL function.

```
@prefix fno: <https://w3id.org/function/ontology#> .
@prefix fnoc: <https://w3id.org/function/vocabulary/composition#> .
@prefix fns: <http://example.com/functions#> .

fns:ETL a fno:Function ;
    fno:expects ( fns:fpathMappingParameter ) ; fno:returns ( fns:returnOutput ) .

fns:ETLComposition a fnoc:Composition ;
    fnoc:composedOf
      # map the input of fns:ETL (fpathMappingParameter) to the input of fns:generateRDF
      [ fnoc:mapFrom [ fnoc:constituentFunction fns:ETL ;
                       fnoc:functionParameter fns:fpathMappingParameter ] ;
        fnoc:mapTo   [ fnoc:constituentFunction fns:generateRDF ;
                       fnoc:functionParameter fns:fpathMappingParameter ] ] ,
      # map the output of fns:generateRDF (fpathOutputParameter) to the input of fns:publish
      [ fnoc:mapFrom [ fnoc:constituentFunction fns:generateRDF ;
                       fnoc:functionOutput fns:fpathOutputParameter ] ;
        fnoc:mapTo   [ fnoc:constituentFunction fns:publish ;
                       fnoc:functionParameter fns:inputRDFParameter ] ] ,
      # map the output of fns:publish (returnOutput) to the output of fns:ETL
      [ fnoc:mapFrom [ fnoc:constituentFunction fns:publish ;
                       fnoc:functionOutput fns:returnOutput ] ;
        fnoc:mapTo   [ fnoc:constituentFunction fns:ETL ;
                       fnoc:functionOutput fns:returnOutput ] ] .
```

Listing 2: FnO Composition ETL workflow description where the output of the first task is linked to the input of the second task

We aligned FnO compositions to standardized provenance information using P-PLAN, complementary to the existing alignment between FnO and PROV-O [15]. By aligning with P-PLAN we benefit from existing work that provides interoperability with several prominent workflow systems [14].

A fno:Function can be linked to one or more fno:Parameter(s) and one or more fno:Output(s). Furthermore, it can be considered as a planned execution activity (p-plan:Step), using (fno:expects/p-plan:hasInputVar) and producing (fno:returns/p-plan:hasOutputVar) variables (p-plan:Variable). A fno:Composition combines one or more fno:Functions, and describes how parameters and outputs are connected. It can be considered as a p-plan:Plan, which is composed of smaller steps (fno:Function(s)). The execution of a fno:Function can be considered as a p-plan:Activity. Listing 3 shows a snippet of how to construct P-PLAN descriptions from FnO compositions. The full snippet is available online [2].

```
PREFIX p-plan: <http://purl.org/net/p-plan#>
PREFIX fnoc: <https://w3id.org/function/vocabulary/composition#>
```

[2] https://gist.github.com/gertjandemulder/9ed4b3d21f10fe62e3689473453fffd9

```
CONSTRUCT {
  ?c a p-plan:Plan .
  ?x p-plan:isStepOfPlan ?c; p-plan:hasInputVar ?xIn;  p-plan:hasOutputVar ?xOut.
  ?y p-plan:isPrecededBy ?x;
     p-plan:isStepOfPlan ?c; p-plan:hasInputVar ?xOut; p-plan:hasOutputVar ?yOut.
WHERE {
    ?c a fnoc:Composition; fnoc:composedOf ?item.
    ?item fnoc:mapFrom ?compositionSource ; fnoc:mapTo  ?compositionTarget .
    ?compositionSource fnoc:constituentFunction ?x .
    ?compositionTarget fnoc:constituentFunction ?y .
    OPTIONAL { # case: f.in -> f.in
      ?compositionSource fnoc:functionParameter ?xIn .
      ?compositionTarget fnoc:functionParameter ?yIn .
    }
    OPTIONAL { # case: f.out -> f.in
      ?compositionSource fnoc:functionOutput ?xOut .
      ?compositionTarget fnoc:functionParameter ?yIn .
    }
    OPTIONAL { # case: f.out -> f.out
      ?compositionSource fnoc:functionOutput [] .
      ?compositionTarget fnoc:functionOutput ?yOut .
    }
  }
```

Listing 3: Pseudo-SPARQL query for constructing the precedence relations in P-PLAN from the CompositionMappings in FnO.

4. Use case

In this section we discuss POSH (Predictive Optimized Supply Chain): a motivating use case showcasing the need for an interoperable KG construction workflow.

POSH is an imec.icon research project in which methods and software solutions are researched that leverage data to optimize integrated procurement and inventory management strategies. A data integration and quality framework is necessary to increase the accuracy and reliability of supply chain data that has been collected from heterogeneous data sources (suppliers, customers, service providers, etc.). To this end, a KG is generated from the heterogeneous supply chain data and consequently exposed through a triple store endpoint. This enables our partners to take advantage of running queries against a uniform data model without being burdened with heterogeneous sources from which it constitutes, and focus on the designing algorithms for optimizing the supply chain.

Within POSH, we developed a semantically-enhanced knowledge integration framework that uses various data repositories and external (meta)data to provide a clear overview of the current state of the supply chain and the necessary inputs for the prediction, optimization and decision support methods. To iteratively accommodate for changing requirements and constraints, an implementation-independent workflow system was needed. Within POSH, we applied our method to provide workflow system flexibly enough to adapt to different technology stacks.

5. Demonstration

We created a proof-of-concept Function Handler that automatically executes FnO Composition descriptions using different implementations, available at https://github.com/FnOio/function-handler-js/tree/kgc-etl. Furthermore, we provide tests[3] in which we verify the execution sequence of an FnO Composition, and demonstrate the interoperability through function compositions that resemble a KG construction workflow in which the RDF-generation task can be implemented by different tools.

A fno:Function can represent anything executable, independent from the context in which it can be executed, whether it is executed by an implementation in a specific programming language (Java, JavaScript, Python, etc.), a command line tool, or Web service, etc. Not only does this allow for interoperability across execution contexts, it also increases the flexibility by which a function can be executed, hence, allowing users to develop custom functions when developing a composition, consequently adding to the expressiveness in which workflows can be defined using FnO. Although scalability is beyond the scope of this paper, we argue that the functional paradigm, which is implicitly enforced by FnO, enables execution engines to exploit existing data-processing frameworks (e.g. Flink and Spark).

6. Conclusion

Declarative function descriptions, and compositions thereof, allow us to define interoperable and reproducible workflows. Interoperable as they are decoupled from the execution environment, where explicit semantics allow for the unambiguous definition of inputs, outputs and implementations. Alignment with PROV allows for a reproducible workflow as both tasks and execution details are provided, which enables to exactly determine which functions were applied throughout the execution of the workflow. Alignment with P-PLAN increases interoperability between workflow systems based on P-PLAN. For example, WINGS[4] is semantic workflow system for computational experiments that uses P-PLAN as an internal representation and can execute its workflows on different existing execution engines. By extending existing P-PLAN alignments [14], we can make use of the CWL ecosystem to execute our workflows described in FnO, without diminishing our currently introduced interoperability and reproducibility. We argue that we could exploit these bridges to translate workflows decriptions in FnO to other systems, such as CWL. This also decreases our dependency on self-introduced projects such as the Function Handler.

For future work, we can gradually increase the transparency of a mapping process as we can interpret a mapping language as a way to describe compositions of transformation tasks. By representing, e.g., a Triples Map in RML as a composition of data and schema transformation tasks, we can provide insights in what a mapping does, and in what order. These insights could help to provide optimization strategies to such kind of engines.

[3] https://github.com/FnOio/function-handler-js/blob/kgc-etl/src/FunctionHandler.test.ts
[4] https://www.wings-workflows.org/

References

[1] C. Goble, et al., FAIR computational workflows, Data Intelligence (2020). doi:10.1162/dint_a_00033.

[2] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent Function Reuse, Future Generation Computer Systems (2020). doi:10.1016/j.future.2019.10.006.

[3] A. Dimou, R. Verborgh, M. V. Sande, E. Mannens, R. V. de Walle, Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval, in: Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15, 2015. doi:10.1145/2814864.2814873.

[4] D. Van Assche, G. Haesendonck, G. De Mulder, T. Delva, P. Heyvaert, B. De Meester, A. Dimou, Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: Web Engineering, Lecture Notes in Computer Science, Springer, 2021. doi:10.1007/978-3-030-74296-6_26.

[5] U. Şimşek, et al., Knowledge Graph Lifecycle: Building and Maintaining Knowledge graphs, in: Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), 2021.

[6] S. Jozashoori, M.-E. Vidal, MapSDI: A scaled-up semantic data integration framework for knowledge graph creation, in: On the Move to Meaningful Internet Systems: OTM 2019 Conferences, 2019. doi:10.1007/978-3-030-33246-4_4.

[7] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Computer Science - Research and Development (2009). doi:10.1007/s00450-009-0057-9.

[8] F. da Silva, et al., Workflows Community Summit: Bringing the Scientific Workflows Community Together, Technical Report, 2021. doi:10.5281/ZENODO.4606958.

[9] A. Barker, J. van Hemert, Scientific workflow: A survey and research directions, in: Parallel Processing and Applied Mathematics, 2008. doi:10.1007/978-3-540-68111-3_78.

[10] Y. Gil, D. Garijo, M. Knoblock, A. Deng, R. Adusumilli, V. Ratnakar, P. Mallick, Improving Publication and Reproducibility of Computational Experiments through Workflow Abstractions, in: K-CAP Workshops, 2017.

[11] M. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanić, H. Ménager, S. Soiland-Reyes, C. Goble, Methods included: Standardizing computational reuse and portability with the common workflow language, arXiv.org (2021).

[12] A. E. Ahmed, et al., Design considerations for workflow management systems use in production genomics research and the clinic, Scientific Reports (2021). doi:10.1038/s41598-021-99288-8.

[13] D. Garijo, Y. Gil, The P-PLAN Ontology, Technical Report, Ontology Engineering Group, 2014. URL: http://purl.org/net/p-plan#.

[14] D. Garijo, Y. Gil, O. Corcho, Towards workflow ecosystems through semantic and standard representations, in: 2014 9th Workshop on Workflows in Support of Large-Scale Science, 2014. doi:10.1109/works.2014.13.

[15] B. De Meester, A. Dimou, R. Verborgh, E. Mannens, Detailed Provenance Capture of Data Processing, in: Proceedings of the 1st Workshop on Enabling Open Semantic Science, 2017.