Phenoflow: Portable Workflow-based Phenotype Definitions

Martin Chapman[1], Luke V. Rasmussen[2], Jennifer A. Pacheco[2], Vasa Curcin[1]
[1]King's College London, London, United Kingdom; [2]Northwestern University, Chicago, Illinois, USA

Abstract

Clinical phenotyping is an effective way to identify patients with particular characteristics within a population. In order to enhance the portability of a phenotype, it is often defined abstractly, with users expected to realise the phenotype computationally before executing it against a local dataset. However, complex definitions, which also provide little information about how best to implement a phenotype in practice, mean that this process is often not easy. To address this issue, we propose a new multi-layer model for a phenotype definition, which is realised as a workflow, and can be combined with different implementation units in order to produce a computable form. A novel authoring architecture, Phenoflow, supports the generation of these structured definitions. To illustrate the utility of our approach, we re-author a diabetes phenotype definition, and then compare its portability to the original definition, in the context of a population of 26,406 patients at Northwestern University.

Introduction

Learning Health Systems require high-quality, routinely collected electronic health record (EHR) data to drive analytics and research, and translate the outputs of novel techniques such as machine learning into patient care and service improvement. To achieve this, the data used for research need not only be of high-quality, but methods associated with its use need be transparent and reproducible to ensure that any findings can be validated by the research community and generalised to other populations. At the core of this challenge is the ability to reliably identify clinically equivalent research-grade patient cohorts capturing a particular disease, sets of comorbidities, medical histories, a demographic profile or any other relevant patient-specific information – a process known as clinical phenotyping.

The popularity of EHR data for research has increased the documentation and sharing of clinical phenotypes derived from research datasets in order to stimulate reuse, reduce variation in phenotype definitions across data sources, and ultimately simplify and support the identification of clinically equivalent populations for research and healthcare applications. The reuse of existing phenotype definitions requires the ability to discover and access curated and validated phenotypes. Pioneering efforts in building standardised phenotype repositories, such as the Phenotype Knowledge Base (PheKB), CALIBER, Million Veterans Program (MVP), and All-of-Us consortium have achieved notable success within their research programmes, with thousands of registered usages.

In an attempt to ensure the portability of a phenotype across multiple research use cases, the logic that comprises a phenotype definition is often represented abstractly within these repositories, where it is structured as, for example, a data flow diagram (e.g., Figure 1). This abstract representation is designed to guide the development of a computable form of the phenotype, such as an executable script or a data pipeline, for a particular use case (e.g. Figure 2). However, in practice, the portability of these definitions is often low: complexity, both in terms of structure and terminology, in the abstract definition makes it hard to interpret in order to produce a computable form, and the technical skill burden on the computable phenotype author is high, as the abstract nature of the definition means that little is communicated about the realisation of the phenotype in practice.

Background

The issue of translating an abstract phenotype definition into a computable form is well recognised within the research community. For example, within the electronic Medical Records and Genomics (eMERGE) Network, phenotype representation for dissemination and publication was typically done using a narrative description of the pseudocode needed to implement the phenotype, with accompanying data flow diagrams. Each institution within the eMERGE Network would take the narrative description and translate it into executable code that would run against their local data warehouse. In response to this inefficiency, over time, this approach has progressed to explore pipeline-based executable representations, such as those using the Konstanz Information Miner (KNIME) analytics platform, which allows the definition of the computable form in a graphical manner.

More recently, the eMERGE Network has adopted a common data model (CDM) - the Observational Medical Outcomes Partnership (OMOP) CDM - to facilitate the representation and dissemination of phenotype algorithms[1]. This

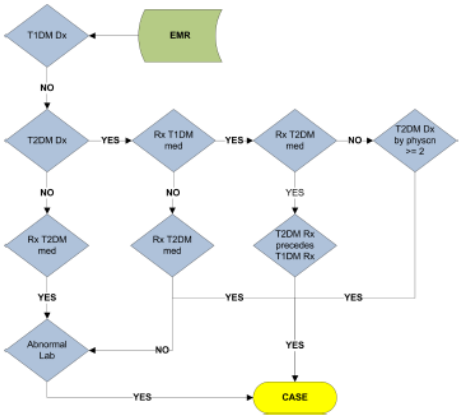

Figure 1: Data flow for defining T2DM patients in an EHR system

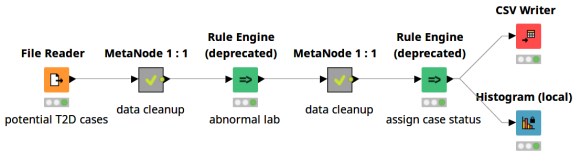

Figure 2: Implemented data pipeline for defining T2DM patients in an EHR system

has allowed the graphical authoring of phenotypes using the Atlas authoring tool. This provides a human-readable representation of the logic, with the benefit of being stored in a format that may be automatically converted to an executable format across multiple database systems at different organisations. This aims to address the issues associated with manually translating an abstract definition into a compatible form and has facilitated the rapid sharing and execution of phenotypes, but as the OMOP CDM is not globally adopted (although it has seen wide growth and adoption in recent years), the representations are therefore not fully portable to other CDMs or local data models[2].

In an attempt to address this issue, the Phenotype Execution and Modelling Architecture (PhEMA) project has proposed the use of a graphical authoring environment that can be used to generate a higher-level, standardised representation of the phenotype logic[3]. Initial work utilised the Quality Data Model (QDM), with more recent development adopting the Clinical Quality Language (CQL). A key aspect of PhEMA's approach is the use of *translators* to take the higher-level representation (QDM or CQL), and convert it into an executable format that may run against a particular CDM. For example, the approach of converting QDM into an executable KNIME workflow allowed that KNIME representation to still be customised for local execution[4,5]. However, while this also aims to solve the issue of developing a computable phenotype based on an abstract representation, the translators available are often specific to an implementation format, such as KNIME. Moreover, the KNIME workflows generated are still tightly coupled to a single environment, such as i2b2 or the local data warehouse, and changes to the original phenotype definition requires recreating the KNIME workflow, and manually re-applying any configurations.

Methods

In order to address the difficulty of deriving computable forms from phenotype definitions, we sought to expand upon the previous work within this area by proposing a novel phenotype definition model, which aims to address complexity, and more explicitly define the structure of computable forms.

The formulation of the proposed model was based on the experiences of initiatives such as UK eScience and US Cyberinfrastructure programmes[6], which developed models for orchestrating and coordinating their computational tasks. In addition, the functional (re-)modelling of different phenomena in a number of different domains was used as the basis for the development of the new model. In particular, work in *hierarchical modelling*; the representation of a phenomena at different levels of abstraction[7], e.g. in bioinformatics software architectures[8]. Finally, the authors themselves have developed a number of different models as part of prior studies, including work on complex systems[9], semantics for hierarchical composition[10] and phenotyping from large scale EHR repositories[11]. This work was used to support the development of a new model.

In order to evaluate the utility of our new *structured definition* model, we first re-authored an existing definition for Type 2 Diabetes Mellitus (T2DM) according to this model (https://github.com/kclhi/t2dm), and then used this definition to produce a new computable form. We then compared the portability of the two definitions using the *Knowledge conversion, clause Interpretation, and Programming* (KIP) phenotype portability scoring system[12]. The existing def-

inition, with corresponding (KNIME) implementation, was obtained from PheKB (https://phekb.org/phenotype/type-2-diabetes-mellitus), and is shown in Figures 1 and 2.

As a part of our evaluation, we employed a dataset consisting of a cohort of 26,406 possible T2DM patients, taken from the Northwestern Medicine Enterprise Data Warehouse (NMEDW). This dataset included a subset ($n = 23$ cases) that had previously undergone manual chart review and acted as the gold standard. We also used this dataset to verify that the structured definition still captured the required phenotype logic, resulting in a consistent implementation. To do this, the new computable form was executed against the publicly available example data on PheKB, and against the NMEDW cohort. This is possible because it uses the same data input format as the original KNIME workflow implementation, and similarly creates an output file in the same structure. The results of this execution were compared to the results the original KNIME implementation, and the same exact cases and controls were found for both the example and NMEDW datasets.

Structured Phenotype Definition

The structured phenotype definition model developed consists of a set of *layers*: an abstract layer, expressing the logic of a phenotype through a set of simple sequential, potentially nested steps, each of which is annotated with multiple descriptions; a functional layer, expressing information about the form of the data passed between the operations within the abstract layer; and a modular computational layer, defining an environment for the execution of one or more implementation units (e.g. a script, data pipeline module, etc.) for each step in the abstract layer.

To express a structured phenotype definition in practice, we rely on the use of *workflows*. We note that the workflow visual paradigm and associated workflow engines are well-equipped to represent all three of the layers in our proposed model, making it an ideal implementation target[13]. A template workflow is thus also developed, specifying the components used to define each layer. In addition, to support the design of structured phenotype definitions, we develop a service architecture, *Phenoflow*, which is able to generate workflows expressed in the *Common Workflow Language* (CWL) as the result of an author interacting with a graphical front-end in order to define each layer of their definition. CWL is chosen due to its provision of the required workflow components, including its support for visualisation tools, as well as its support for namespaces, and technical integration with container technologies such as *Docker*.

In the following sections, we expand upon the detail of each layer in a structured phenotype definition, and present the workflow template that is used to express these layers in practice. We then provide details of the Phenoflow authoring architecture.

Abstract

```
class: Workflow
...
steps:
  '[stepId]':
    ...
    run: [toolId|nestedWorkflowId]
```

Figure 3: Structured phenotype definition workflow template: abstract layer

```
class: CommandLineTool
$namespaces:
  s: [namespaceURI]
...
doc: [stepDescription]
id: [stepId]
...
s:type: [stepOntologyType]
```

Figure 4: Structured phenotype definition tool template: abstract layer

Like traditional definitions, the abstract layer of a structured phenotype definition also holds the logic of the phenotype. However, the abstract layer of a structured phenotype definition is defined by two key features. Firstly, this layer consists of a discrete number of sequential steps, each of which defines a single operation against a target dataset. However, steps may also be grouped, allowing for their functionality to be summarised by a single parent step. The second feature of this layer is a multi-dimensional description of each step, which consists of an ID, summarising the purpose of the step using relevant clinical terminology; a longer description of the step, offering a non-technical description of the logic of the step; and a categorisation of the logic of the step as an entity within a given concept ontology.

The workflow components required to express this information are defined in the (CWL) templates show in Figures 3 and 4, where the former is a *workflow*, and the latter is a connected *tool*. If a step in a workflow refers to a single

operation, a connected tool is used; if it refers to a group of steps (i.e. it is a parent step), a connected *nested workflow* is used. These workflow variables are used as the basis for generating a visual representation of the abstract layer, thus presenting the layer in a format more commonly seen.

Functional

```
1  class: Workflow
2  ...
3  inputs:
4    [workflowInputId]:
5      id: [workflowInputId]
6      type: [workflowInputType]
7      doc: [workflowInputDescription]
8  outputs:
9    [workflowOutputId]:
10     id: [workflowOutputId]
11     outputBinding:
12       glob: '*.[
             workflowOutputExtension]'
13     outputSource: [stepId]/[
           stepOutputId]
14     type: [workflowOutputType]
15 steps:
16   '[stepId]':
17     in:
18       [stepInputId]:
19         id: [stepInputId]
20         source: [workflowInputId]
21     out:
22     - [stepOutputId]
23     run: [toolId|nestedWorkflowId].cwl
```

Figure 5: Structured phenotype definition workflow template: functional layer

```
1  class: CommandLineTool
2  ...
3  id: [stepId]
4  inputs:
5  - id: [stepInputId]
6    doc: [stepInputDescription]
7    type: [stepInputType]
8  outputs:
9  - id: [stepOutputId]
10   doc: [stepOutputDescription]
11   type: [stepOutputType]
12   outputBinding:
13     glob: '*.[stepOutputExtension]'
14 ...
```

Figure 6: Structured phenotype definition tool template: functional layer

The functional layer of a structured phenotype definition augments the information held in the abstract layer by adding metadata information about the inputs and outputs of each step, thus providing a specification that any implementation must satisfy, regardless of the exact technology used. This is an application of a functional programming paradigm[14] and is realised using the components shown in Figures 5 and 6. These include the inputs and outputs to and from each step (e.g. Figure 5, Line 18); the inputs passed to the first step, and the outputs received from the last by the workflow (e.g. Figure 5, Line 4); and the inputs sent to, and the outputs from, a tool by each step (e.g. Figure 6, Line 5). The input and output metadata also includes relevant identifiers and syntactical commitments (data formats or file types).

Computational

```
1  class: CommandLineTool
2  ...
3  arguments:
4  - [applicationInputs]
5  baseCommand: [applicationBaseCommand]
6  ...
7  id: [stepId]
8  inputs:
9  - id: [stepInputId]
10   inputBinding:
11     prefix: -[applicationPrefix]=
12     separate: [separation]
13     valueFrom: [valueFromLogic]
14   ...
15 ...
16 requirements:
17   DockerRequirement:
18     dockerOutputDirectory: [applicationOutputDirectory]
19     dockerPull: [applicationImageLocation]
20 ...
```

Figure 7: Structured phenotype definition tool template: computational layer

The computational layer of a structured phenotype definition specifies the presence of one or more implementation units for each (nested) step in the abstract (and functional) layers. This includes information about the execution environment used to run the implementation unit, and how the unit is linked to that environment. An implementation unit itself is connected to a step in the computational layer by referencing it as an additional input.

In a workflow, the tools connected to a step, or a nested step, enable the specification of this information (Figure 7), such as the container image that comprises the execution environment (Figure 7, Line 19) and the prefix used to inform the environment (within a container) of the location of an implementation unit (Figure 7, Line 11). Multiple tools referencing the same step identifier provide support for multiple implementation units.

Architecture

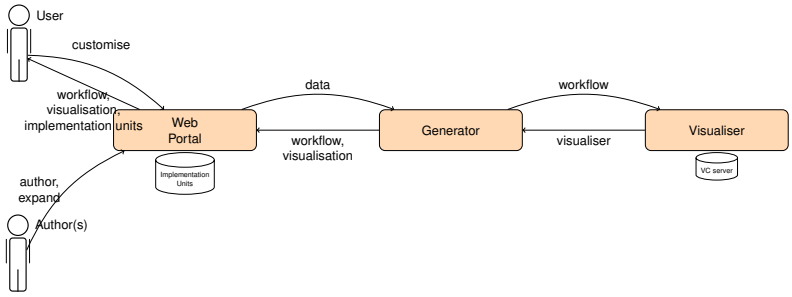

Figure 8: The services that constitute the *Phenoflow* architecture.

In order to author a structured phenotype definition in practice, we do not anticipate authors instantiating the template that realises our model directly. Instead, we define a service architecture, *Phenoflow*, which provides a number of services to enable the instantiation of these workflows automatically. An overview of this architecture is given in Figure 8, and its services are expanded upon in the following sections.

Web portal

The primary service in the Phenoflow architecture is a web portal, through which definitions are graphically authored at the abstract and functional layers, and the computational layer is generated via the provision an implementation unit for each step in the upper layers. Following this, other authors are able to introduce new computational layer permutations, by supplying alternative implementation units for the same definition. Users can then obtain a computable form of a phenotype by accessing its definition and its associated implementation units.

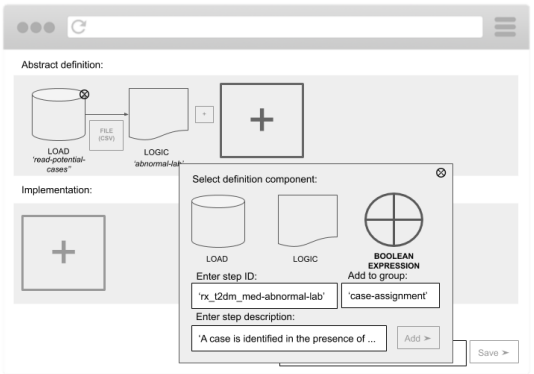

Figure 9: Visually defining the abstract and functional layers of a definition.

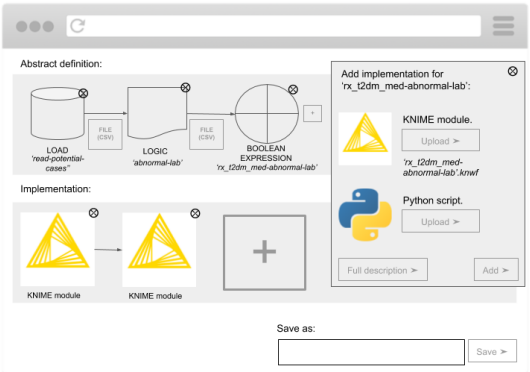

Figure 10: Providing an implementation unit for the third step in the abstract and functional layers, in order to generate a step in the computational layer and store this unit.

Authoring

When authoring a definition, a researcher expresses the steps of their new definition at the abstract and functional layers by selecting the type of entity (broadly based on the axioms of the PhEMA authoring tool (PhAT)[15]) and then

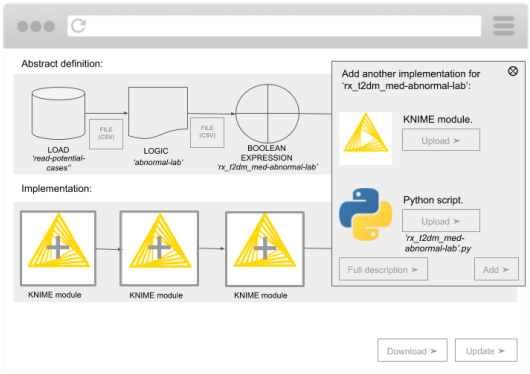

Figure 11: Adding additional implementation units for an existing step.

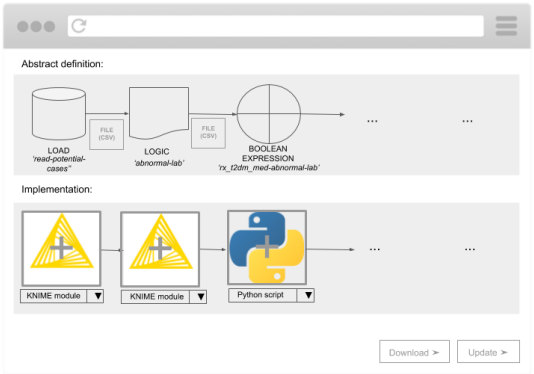

Figure 12: Customising a computable phenotype for local use.

by labelling, describing and, if they wish to, grouping this entity, as well as describing inputs and outputs. This process is represented in Figure 9, where a user is in the process of defining a new boolean expression, within their abstract layer, having already defined an initial data read, another piece of logic, and the fact that a file is passed between these two steps.

Following the specification of one or more steps in the abstract and functional layers, the researcher graphically connects each step to an implementation unit (e.g. a Python script, or a KNIME module; their choice across the steps does not have to be homogeneous), which they supply to the portal, in order to generate the computational layer. This process is represented in Figure 10, where the author is uploading the module of KNIME workflow as the computable counterpart of their priorly defined boolean expression step.

Expanding

If another researcher wishes to later supply an alternative implementation unit for any of the existing units, thus introducing a permutation of the computational layer, they can do so, and this process is represented in Figure 11. Here, another author has accessed a previously authored definition, and is in the process of adding an alternative implementation for the third step in the computational layer; previously implemented as a KNIME module, the second author is now uploading a Python realisation of the same abstract boolean expression.

Customising

Given the potential for multiple permutations of the computational layer, and associated implementation units, when accessing the definitions authored by others, a user is able to pick the permutation they wish to use in order to generate a computable phenotype for local use. This process is represented in Figure 12, where a user is selecting, from the stored implementation units, the exact structure of the computable phenotype; they have chosen a permutation that mixes KNIME and Python implementation units.

Generator

Once a user has customised the implementation of their definition, the information elicited by the web portal is used to populate our CWL template. To do this in practice, the web portal sends this information to the generation service in the Phenoflow architecture, which is backed by *python-cwlgen* (https://github.com/common-workflow-language/python-cwlgen). An example of such an instantiated workflow (tool) template, based on the information gathered in a scenario like the one shown in Figures 9 to 12, can be found in later sections.

Visualiser

Prior to responding to the web portal, the generator contacts the visualisation service in the Phenoflow architecture, which is backed by *cwlviewer* (https://github.com/common-workflow-language/cwlviewer), and supplies it with a copy of the generated workflow, via a local versioning server. In response, the visualisation service sends back a visualisation of the abstract and functional layers expressed in the supplied workflow, and this is combined with the

	Knoweldge	Clause	Programming	Total
T2DM original	1	1	2	4
T2DM structured	0	1	0	1

Table 1: KIP portability scores assigned to the original T2DM phenotype definition and its structured counterpart.

generated workflow and sent back to the web portal by the generator. An example of a workflow visualisation, based on the information gathered in a scenario like the one shown in Figures 9 to 12, can be found in later sections. Once the web portal has a copy of the instantiated workflow (the structured definition), and the visualisation counterpart, it combines this with the locally stored implementation units, referenced in the workflow, and packages them as a download for the user to access locally as a computable phenotype.

Results

The following sections present our KIP scores from the comparison of the structured and original T2DM phenotype definitions, and associated reasoning. These scores are summarised in Table 1. The KIP assigns higher scores to phenotypes that are less portable.

Knowledge conversion

The first aspect of the KIP scoring system relates to the clinical knowledge required to develop a computable phenotype from its definition. In the original phenotype definition, we note the use of some medical terminology (e.g., 'T2DM Rx precedes T1DM Rx', Figure 1). Supplementary information about the meaning of this terminology is provided, but not within the definition itself, and in a different form (as an additional written document). This follows the example provided within the KIP model to assign a score of 1 for knowledge conversion for the original definition.

```
1 class: CommandLineTool
2 $namespaces:
3   s: https://phekb.org/
4 ...
5 doc: A case if identified in the presence of an abnormal lab value (defined as one of three different
      exacerbations in blood sugar level) and if medication for this type of diabetes has been prescribed.
6 id: rx_t2dm_med-abnormal-lab
7 ...
8 s:type: boolean-expression
```

Figure 13: Extract from T2DM structured phenotype definition abstract layer

An extract from the abstract layer of the structured T2DM phenotype is shown in Figure 13. When examining this layer, we encounter terminology similar to the terminology use in the original definition (e.g., 'rx_t2dm_med-abnormal-lab'), however the impact this has on portability is lessened in two key ways. Firstly, additional information about the meaning of the terminology is provided within the abstract layer itself, in the description field of each step, ensuring that any medical terminology is supplemented by a longer, more accessible, description (e.g., Figure 13, Line 5). Secondly, the classification of each step as a type of operation from a pre-defined (PheKB) ontology ensures that even in the presence of medical terms, basic understanding about the logic of a step can still be extracted. For example, the classification of a step containing a case assignment rule as a boolean expression ensures that the use of medical terminology does not obscure its logic (Figure 13, Line 8). Based upon these factors, the KIP system assigns a value of 0 for this aspect.

Clause interpretation

The second aspect of the KIP scoring system aims to identify any ambiguity in the logical clauses found in a phenotype definition, that may result in inconsistencies when realising this logic computationally. The existing T2DM phenotype definition (Figure 1) uses long conditional clauses (represented graphically), however the logic still has a clear interpretation. This leads to the attribution of a further KIP score of 1.

A visualisation containing the abstract layer of the re-authored T2DM definition is shown in Figures 14 and 15. Figure 15 shows a grouping of *case assignment* rules as a nested workflow, referenced by the parent step shown in orange in Figure 14. This group contains a series of steps that are evaluated in sequence, each of which contains an individual boolean expression, such as the one defined in Figure 13. This breaks down the long clauses seen in the original

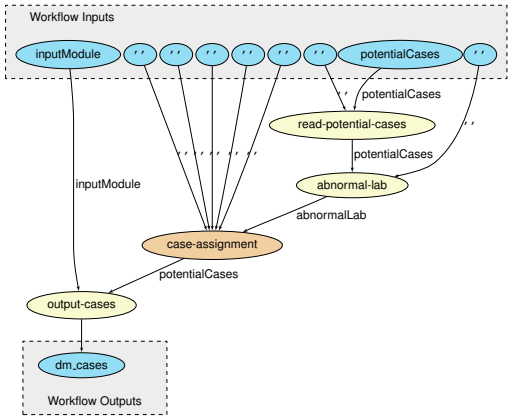

Figure 14: Visualisation of T2DM structured phenotype definition abstract layer.

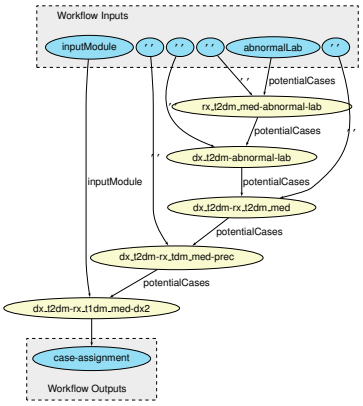

Figure 15: Visualisation of T2DM structured phenotype definition abstract layer (nested).

abstract layer, while retaining a clear interpretation. Moreover, the use of a group (nested workflow) itself, furthers this simplicity by allowing for the overall role of these steps to be more easily identified within the abstract layer. Despite this simpler overall structure, at the same time the additional expressions within the steps introduce different complexity. For these reasons, the same KIP score of 1 is assigned to this aspect under the structured definition.

Programming

The final aspect of the KIP scoring system relates to the programmatic complexity of implementation. The structure of the original T2DM definition suggests a moderate level of programming expertise required to produce a computable form (e.g. the requirement for a script to be produced in order to realise the stated logical conditions). This is compounded by the fact that there is little instruction that can be extracted from the definition on how to develop the computable form in practice. This results in a disconnect between the two, which reduces the intelligibility of the latter. For example, the case assignment logic, seen in Figure 1, while defined as separate operations in the definition is obscured in a single node in the computable form (Figure 2), making the correspondence between the two unclear. This reduced intelligibility makes it harder to reuse, or modify, the provided implementation in a new use case. As a result, a KIP score of 2 is awarded.

```
$namespaces:
  s: https://phekb.org/
arguments:
- --data /home/kclhi/.eclipse -reset -nosplash -nosave -application org.knime.product.KNIME_BATCH_APPLICATION
baseCommand: /home/kclhi/knime_4.1.1/knime
class: CommandLineTool
doc: A case is identified in the presence of an abnormal lab value (defined as one of three different exacerbations in blood sugar level) AND if medication for this type of diabetes has been prescribed.
id: rx_t2dm_med-abnormal-lab
inputs:
- doc: KNIME implementation unit
  id: inputModule
  inputBinding:
    prefix: -workflowFile=
    separate: false
  type: File
- doc: Potential cases of this type of diabetes.
  id: potentialCases
  inputBinding:
    prefix: -workflow.variable=dm_potential_cases,file://
    separate: false
    valueFrom: '$(inputs.potentialCases.path),String'
  type: File
outputs:
- doc: Output containing patients flagged as having this case of diabetes.
  id: output
  outputBinding:
    glob: '*.csv'
  type: File
requirements:
  DockerRequirement:
    dockerOutputDirectory: /home/kclhi/.eclipse
    dockerPull: kclhi/knime:amia
s:type: boolean-expression
```

Figure 16: Extract (tool) from T2DM structured phenotype definition.

In contrast, the requirement for a distinct implementation unit for each step in the abstract layer, introduced by the computational layer of a structured definition, each of which responds to the inputs, and produces the outputs, specified in the functional layer, provides a clear template for development. This lessens the implementation burden, by either structuring new development, or allowing existing implementation units, which may have been developed locally, to be reused in order to produce the computable form of a definition.

In addition, a computable phenotype produced on the basis of a structured definition is naturally more intelligible, as it holds a greater correspondence with the description. For example, in the case of T2DM, because each step in the definition, such as the one shown in Figure 16, must be connected to an individual implementation unit, the case assignment logic is no longer obscured, as it was in the original computable form. As a result of this increased intelligibility, these computable phenotypes are more transparent, and thus reusable and more easy to modify, lessening the implementation burden on future developers. Moreover, assuming multiple implementation units for the same abstract step, which can be easily swapped in and out owing to the modularity brought in the structured form, a user is more likely to find a unit written in a technology they are comfortable with, and can thus edit, again reducing the implementation burden. For example, our structured T2DM definition references both KNIME and Python implementation units. This is perhaps the most notable advantage above other solutions, as structured phenotype definitions are not tightly coupled to any singular CDM or technology.

Finally, we note the integration of container technology, enabling any existing implementation units, or newly developed code, to be swiftly executed within a self-contained execution environment that is automatically acquired, rather than requiring manual installation. For example, we develop a custom KNIME Docker image, that can be referenced within the computational layer. All of these factors result in the attribution of a KIP score of 0 for this aspect.

Conclusion and Future work

In this paper, we introduce a multi-layer model for the definition of a phenotype, and an associated architecture, *Phenoflow*, which is used to generate phenotype definitions structured under this model as workflows, which can later be executed along with associated implementation units. In contrast to existing work that aims to autonomously convert abstract definitions into a computable form, this complementary approach aims to be more widely applicable by assisting with the manual phenotype translation process: increasing the intelligibility of abstract definitions by supplementing the use of clinicial terminology and simplifying the representation of logical structures; and increasing the intelligibility of existing computable forms, enabling editing and reuse, as well as guiding novel implementation. Overall, we note a number of improvements to portability when a T2DM phenotype is structured using our representation model, under a defined scoring system, with the exception of clause interpretation, where the fact that long clauses are (necessarily) replaced with individual steps, introduces different complexity with equivalent effect.

Future work will include expanding the evaluation of portability brought by the structured phenotype definition across additional phenotypes, including a focus on diverse implementations across data models. In addition, work will focus on developing libraries of implementation units, to be made available to researchers wanting to customise an existing computable phenotype for their research task. Such libraries will operate within phenotype repositories such as PheKB and CALIBER. Furthermore, the abstract phenotype forms will also be used to develop search capabilities for such phenotype repositories using similarity metrics developed. Finally, helper tools will be developed for common tasks such as automated file type conversions to assist researchers in running published phenotype definitions, e.g. converting a CSV to a lightweight SQL table, so that an SQL script can be transparently executed against the data within an individual step.

References

[1] George Hripcsak, Ning Shang, Peggy L. Peissig, Luke V. Rasmussen, Cong Liu, Barbara Benoit, Robert J. Carroll, David S. Carrell, Joshua C. Denny, Ozan Dikilitas, Vivian S. Gainer, Kayla Marie Howell, Jeffrey G. Klann, Iftikhar J. Kullo, Todd Lingren, Frank D. Mentch, Shawn N. Murphy, Karthik Natarajan, Jennifer A. Pacheco, Wei Qi Wei, Ken Wiley, and Chunhua Weng. Facilitating phenotype transfer using a common data model. *Journal of Biomedical Informatics*, 96, 8 2019.

[2] Jiang G Kiefer RC Pacheco JA Adekkanattu P Ancker JS Wang F Xu Z Pathak J Luo Y Rasmussen LV, Brandt P. Considerations for Improving the Portability of Electronic Health Record-Based Phenotype Algorithms. In *AMIA*

Symposium, 2019.

[3] Luke V Rasmussen, Richard C Kiefer, Huan Mo, William K Thompson, Guoqian Jiang, Jennifer A Pacheco, Peter Speltz, Jie Xu, Joshua C Denny, Enid Montague, and Jyotishman Pathak. The Phenotype Execution and Modeling Architecture (PhEMA) - A Standards-Based Composition of Software for Phenotype Algorithm Development. Technical report, Northwestern, 2015.

[4] Huan Mo, Guoqian Jiang, Jennifer A Pacheco, Richard Kiefer, Luke V Rasmussen, Jyotishman Pathak, Joshua C Denny, and William K Thompson. A Decompositional Approach to Executing Quality Data Model Algorithms on the i2b2 Platform. *AMIA Joint Summits on Translational Science*, 2016:167–75, 2016.

[5] Jennifer A Pacheco, Luke V Rasmussen, Richard C Kiefer, Thomas R Campion, Peter Speltz, Robert J Carroll, Sarah C Stallings, Huan Mo, Monika Ahuja, Guoqian Jiang, Eric R LaRose, Peggy L Peissig, Ning Shang, Barbara Benoit, Vivian S Gainer, Kenneth Borthwick, Kathryn L Jackson, Ambrish Sharma, Andy Yizhou Wu, Abel N Kho, Dan M Roden, Jyotishman Pathak, Joshua C Denny, and William K Thompson. A case study evaluating the portability of an executable computable phenotype algorithm across multiple institutions and electronic health record environments. *Journal of the American Medical Informatics Association*, 25(11):1540–1546, 8 2018.

[6] Tony Hey and Anne E. Trefethen. Cyberinfrastructure for e-science. *Science*, 308(5723):817–821, 2005.

[7] Fabrice Bernardi and Jean François Santucci. Model design using hierarchical web-based libraries. In *Proceedings - Design Automation Conference*, pages 14–17, New York, New York, USA, 2002. ACM Press.

[8] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole A. Goble, Matthew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web-Server-Issue):729–732, 2006.

[9] M. Chapman, G. Tyson, P. McBurney, M. Luck, and S. Parsons. Playing hide-and-seek: An abstract game for cyber security. In *ACM International Conference Proceeding Series*, 2014.

[10] V. Curcin, M. Ghanem, and Y. Guo. The design and implementation of a workflow analysis tool. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1926), 2010.

[11] Vasa Curcin, Alex Bottle, Mariam Molokhia, Christopher Millett, and Azeem Majeed. Towards a scientific workflow methodology for primary care database studies. *Statistical Methods in Medical Research*, 19(4):378–393, 2010. PMID: 20442191.

[12] Ning Shang, Cong Liu, Luke V. Rasmussen, Casey N. Ta, Robert J. Caroll, Barbara Benoit, Todd Lingren, Ozan Dikilitas, Frank D. Mentch, David S. Carrell, Wei Qi Wei, Yuan Luo, Vivian S. Gainer, Iftikhar J. Kullo, Jennifer A. Pacheco, Hakon Hakonarson, Theresa L. Walunas, Joshua C. Denny, Ken Wiley, Shawn N. Murphy, George Hripcsak, and Chunhua Weng. Making work visible for electronic phenotype implementation: Lessons learned from the eMERGE network. *Journal of Biomedical Informatics*, 99, 11 2019.

[13] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, and Ian Taylor. Scientific workflows: Past, present and future. *Future Generation Computer Systems*, 75:216–227, 2017.

[14] Richard Bird and Philip Wadler. *An Introduction to Functional Programming*. Prentice Hall International (UK) Ltd., GBR, 1988.

[15] Luke V Rasmussen, Richard C Kiefer, Huan Mo, Peter Speltz, William K Thompson, Guoqian Jiang, Jennifer A Pacheco, Jie Xu, Qian Zhu, Joshua C Denny, Enid Montague, and Jyotishman Pathak. A Modular Architecture for Electronic Health Record-Driven Phenotyping. *AMIA Joint Summits on Translational Science*, 2015:147–51, 2015.