# Towards Model-Driven Semantic Interfaces for Electronic Health Records on Multiple Platforms Using Notation3

William Van Woensel[1[0000-0002-7049-8735]], Samina Abidi[2[0000-0002-7805-6122]], and Syed Sibte Raza Abidi[1[0000-0002-0982-4052]]

[1] NICHE Research Group, Faculty of Computer Science, Dalhousie University, Canada
[2] Community Health and Epidemiology, Faculty of Medicine, Dalhousie University, Canada

**Abstract.** Electronic Health Record (EHR) systems that aim to achieve data interoperability need to conform to established EHR standards, such as HL7 FHIR or openEHR. Manually developing EHR user interfaces (UIs) for the input of standards-compliant health data is not scalable and unsustainable, since this data no longer *only* originates from the health practitioner's desktop—given an increased focus on illness self-management, health data is being increasingly generated from patient-focused web or mobile applications. This has led to the collection of non-standards-compliant data from a variety of health apps, i.e., vitals, symptoms, treatments, and so on—this data is not interoperable and requires significant processing to load into an standards-compliant EHR system. A solution for the collection of interoperable health data is the (semi-)automatic generation of UIs, guided by standards-compliant descriptions of the required health data and their constraints. We present a framework to automatically generate UIs for the collection of interoperable health data, supporting different platforms (i.e., UI formats) and EHR standards. The generated UIs perform input validation and submit a self-contained, semantically annotated EHR record. Currently, we support HTML+RDFa (web) and Yail (mobile) as UI formats, and HL7 FHIR and openEHR as EHR standards. We utilize Notation3 (N3) to implement our UI generation pipeline, a Semantic Web language for decision-making in an open Web environment.

**Keywords:** Electronic Health Records, Semantic User Interfaces, Notation3

## 1 Introduction

To increase self-sufficiency and curb rising healthcare costs, chronic patients are increasingly being encouraged to self-manage their illness—this requires patient educational and motivational resources [1–3] together with tools for self-reporting health data (vitals, drugs, symptoms, and so on) [4–6]. To meet the reporting needs of a specific chronic illness, a tailored patient diary is typically developed and deployed for the web, desktop, or mobile platform. Cross-platform development tools, such as Apache Cordova [7], allow re-using the same code base for multiple platforms. The authors have developed custom patient diaries in the past for Atrial Fibrillation [5] and

developmental delay in the context of the Zika outbreak [6]. Currently, we aim to do the same for Chronic Obstructive Pulmonary Disease (COPD)—however, we want to avoid the large development effort of *yet another* patient diary.

In this vein, the goal of this paper is to study the automatic generation of UIs for collecting interoperable, standards-compliant health data. Indeed, due to (a) different reporting needs of chronic illnesses (e.g., AFib, COPD), (b) the need to support multiple platforms (e.g., desktop, web, mobile), and (c) different EHR standards currently in use, development of UIs for collecting interoperable health data has become an almost combinatorial effort. The good news is that both HL7 FHIR [8] and openEHR [9], two popular open EHR standards, offer structured descriptions of relevant health data and their constraints. Prior work has shown that it is feasible to automatically generate health data interfaces based on openEHR artifacts [10]. We further note that parts of these EHR standards (GLD [11]; CQL [12]), and existing guideline languages (PROForma [13], SDA* [14]), offer the ability to computerize clinical guidelines for decision support, acting on interoperable health data to issue care recommendations [15]. This paper thus fits into our ultimate goal of model-driven (EHR, guideline models) and low-code (reduced development effort) clinical decision support (CDS).

The objective of this particular work is to move towards a generic framework that generates UIs for collecting standards-compliant, interoperable health data, based on structured descriptions of health input data and their constraints. This generic framework aims to support different EHR standards (e.g., HL7 FHIR, openEHR); data collection platforms (desktop, mobile, web), and thus UI formats (e.g., HTML, XUL [16], Yail [17]); and data reporting needs, as per the chronic illnesses. Our generated UI submits a self-contained, semantically annotated EHR record, and performs input validation to the extent supported by the UI format. Both features reduce the need for server-side processing, for instance, needed to convert raw input into a standards-compliant EHR record, and issuing feedback on input data validity. We utilize Notation3 (N3) [18, 19]—a Semantic Web language for decision-making in an open Web environment—to implement our UI generation framework. By supporting if-then style decision making, a scoped negation-as-failure, and quoting of graphs, N3 offers the expressivity needed to implement our UI framework with minimal effort; moreover, we posit that N3 is highly suitable for CDS in general. While we lack the space to detail our N3 implementation, the code is available in an online repository [20].

## 2 Related work

**Open Electronic Health Record standards.** The HL7 FHIR and openEHR are both open standards that are actively being utilized to represent EHR data. However, they have a wholly different focus: FHIR targets standards-based data exchange between systems, which may use any underlying EHR model; whereas openEHR focuses on standardizing the underlying data model within EHR systems. In a nutshell, FHIR defines exchangeable data in terms of *resources*, which are used individually or composed to satisfy a use case, with a built-in extension mechanism to cope with uncommon cases (80% rule: standardizing an element requires 80% of systems to require it). FHIR offers

a serialization in RDF (Turtle syntax) to support inference and shared semantics across multiple standards; this facilitates our work since we utilize N3 in our implementation. OpenEHR follows a two-level modeling approach by Beale [21], where a reference model (RM) defines a complete set of low-level, core EHR terms; higher-level models, called archetypes, represent information structures (such as an blood pressure observation) in terms of the RM. Hence, a system implementing the RM does not need to be re-coded for new observations, but merely requires plugging in new archetypes. OpenEHR does not have an RDF syntax; several works have targeted the mapping of openEHR to ontologies [22, 23]. In this paper, we applied a straightforward conversion to N3 to illustrate our UI generation framework.

**Automated Health UI generation from EHR descriptions.** Schuler et al. [10] showed the feasibility of generating UIs based on openEHR archetypes that are similar to hard-coded interfaces in medical applications. As the number of types in the openEHR RM is relatively small, customized controls can easily be developed for each of them [10]. The authors generated UIs in the Mozilla XML User Interface Language (XUL) from openEHR templates, and hence did not target different platforms or EHR standards. The Medblocks platform [24] aims to convert openEHR templates into Web Components [25], with experimental support for FHIR resources. It is unclear whether different platforms will be supported. A UITemplate Model Specification has been discussed on the openEHR discourse forum [26] that expresses UI metadata and constraints for UI generation for different platforms. However, this effort seems incomplete at this time, and would only target openEHR.

**Notation3 Semantic Web Language.** Notation3 (N3) is a Semantic Web language, originally proposed by Berners-Lee et al. [27], with its formal semantics fleshed out by Arndt et al. [28], and, recently, the topic of a W3C CG [29]. As mentioned in the draft community report [19], N3 is a superset of Turtle, adding (1) if-then style decision making in terms of logic implications and variables, (2) a scoped negation-as-failure, (3) quoting graphs of statements, and (4) set of powerful built-ins. We posit that these features make N3 highly suitable for semantic EHR in general, and we aim to study the utility of N3 for model-driven CDS: (1) if-then reasoning is prevalent in clinical decision making, and (2) it is often useful to check, within an EHR scope, whether a patient does *not* have certain properties (e.g., symptoms). Moreover, there is a well-known impedance mismatch between RDF and EHR [30]: the latter is geared towards recording who did what (e.g., “Dr. X diagnosed patient Y with flu”), but RDF focuses on stating absolute facts (e.g., “patient Y has flu”). Quoted graphs in N3 can thus allow a more accurate image of EHR, describing actors, times, and degrees of belief. We have shown that N3 can support different quoted graph semantics [31]. While we lack the space to detail our N3 implementation, the code is available online [20] and future work will describe it in more detail.

## 3 Artifacts for Health User Interface Generation

Our work targets the automated generation of UIs for interoperable health data collection, based on descriptions of health data and constraints using EHR standards. We

start by describing these EHR artifacts in more detail (Sections 3.1 and 3.2), as well as artifacts needed to support a particular UI format (Section 3.3). Finally, we introduce templates that tie these artifacts together to guide UI generation (Section 3.4).

### 3.1 EHR Health Data Descriptions and Constraints

EHR standards offer high-level descriptions of EHR data in terms of codes from well-known terminologies (e.g., SNOMED-CT [32]), labels in different languages, and constraints such as the expected datatype (e.g., numeric, boolean), relevant units (e.g., BPM, percentage), and valid data ranges. Collectively, these high-level descriptions can be utilized to describe the *data reporting needs* for a (chronic) illness, and thus offer a starting point for automated UI generation [10].

FHIR includes a plan definition resource with activity definitions, which themselves may have observation definitions. Below, we describe constraints related to a particular data reporting need, i.e., scoring cough severity on a scale of 1-10, using FHIR (Turtle syntax):

```
:diary_observations a fhir:PlanDefinition ;
   fhir:PlanDefinition.action ( [
      fhir:PlanDefinition.action.definitionUri :diagnose_cough_wheezing_stridor
   ] … ]

:diagnose_cough_wheezing_stridor a fhir:ActivityDefinition ;
   fhir:ActivityDefinition.title "Is your cough, wheezing or stridor less, the same or
worse than usual?" ;
   fhir:ActivityDefinition.observationResultRequirement [
      fhir:ObservationDefinition.code :code_cough ; …
      fhir:ObservationDefinition.permittedDataType :dt_integer, :1_10_scale
   ] .
:1_10_scale a fhir:Range ;
   fhir:Range.low  [ fhir:Quantity.value 1 ] ;
   fhir:Range.high [ fhir:Quantity.value 10 ] ;
   rdfs:label "(1-10, 1 is least, 5 is same, 10 is worst)" .
```

**Fig. 1.** Example FHIR plan, activity and observation definition[1].

The `:diary_observations` plan definition has a multiple activity definitions, including `:diagnose_cough_wheezing_stridor`, which describes requirements on associated observations: i.e., values must be integers that lie within a 1-10 scale, and any observation will have a specific code (`:code_cough`) associated with it. The `:code_cough` term represents a *codeable concept* that encodes the SNOMED code for "cough (finding)" (not shown). FHIR directly supports the Turtle syntax (Section 2), of which N3 is a superset, meaning that our N3 implementation can directly operate on FHIR data.

OpenEHR targets the holistic modeling of EHR data (Section 2) and descriptions of EHR data thus tend to be far more elaborate. Firstly, an openEHR archetype details all relevant data and constraints for a single observation (e.g., body temperature):

```
archetype (adl_version=1.4; uid=…)
   openEHR-EHR-OBSERVATION.body_temperature.v2
…
definition
```

[1] We refer to our code repository [20] for full code samples.

```
OBSERVATION[at0000] matches { -- Body temperature
   …
   ELEMENT[at0004] occurrences matches {0..1} matches { -- Temperature
      value matches {
         C_DV_QUANTITY <
            list = < ["1"] = <
               units = <"Cel"> magnitude = <|0.0..<100.0|> precision = <|1|> >
                     ["2"] = <
               units = <"[degF]"> magnitude = <|30.0..<200.0|> precision = <|1|>
            > > > } … }
```

**Fig. 2.** Example openEHR archetype[2].

The archetype stipulates that observations must either use the Celsius scale, where values must lie between 0 – 100 with 1 decimal place; or the Fahrenheit scale with its own constraints. Codes such as "at0004" are linked to terminology codes and translations into different languages in the `ontology` component of the archetype (not shown).

Secondly, an openEHR template composes archetypes for a particular purpose (e.g., vital signs report). Since an archetype covers all possibly relevant data (e.g., the BP archetype lists mean arterial pressure, pulse pressure, tilt, and so on), a template often applies constraints on composed archetypes to rule out unneeded data input:

```
<?xml version="1.0" encoding="UTF-8"?>
<template xmlns="openEHR/v1/Template">
   …
   <definition archetype_id="openEHR-EHR-COMPOSITION.encounter.v1" …>
      <Content archetype_id="openEHR-EHR-OBSERVATION.body_temperature.v2" …>
         <Rule path="/data[at0002]/events[at0003]/data[at0001]/items[at0004]">
            <constraint xsi:type="tem:quantityConstraint">
                  <excludedUnits>[degF]</excludedUnits> …
               </constraint>
            </Rule>
         <Rule max="0" path="/data[at0002]/events[at0003]/data[at0001]/items[at0063]"/> …
   … </definition></template>
```

**Fig. 3.** Example openEHR template.

The template includes the body temperature archetype (`content` element), applying a rule on the temperature element that excludes Fahrenheit as a unit (part of `constraint`), and another rule that hides (`max="0"`) body exposure as data input.

The openEHR template can be "flattened" into a more usable "operational template"—a process that injects all data from referenced archetypes and applies any constraints. We utilized the openEHR java-libs[3] library to perform this flattening. Moreover, since openEHR does not directly support an RDF syntax (Section 2), we implemented a visitor object to convert the flattened template into N3.

### 3.2 EHR Data Structures to Report Health Observations

Aside from data reporting needs, an EHR standard also dictates a particular observation structure, i.e., how to concretely encode patient data as observations. In general, this varies greatly between EHR standards: FHIR reports input data using a distinct set

[2] openEHR code samples were obtained from https://ckm.openehr.org/ckm/.
[3] https://github.com/openEHR/java-libs

of predicates and nesting structures, whereas openEHR utilizes a similar structure as found under the *definition* element (Fig. 2). To support multiple EHR standards, our UI generation framework allows defining an *observation structure* per EHR standard and observation type. In particular, an observation structure describes the predicates and structures used to report different types of observations (e.g., numeric scale, quantified value), and is annotated to guide the UI generation process.

Below, we show part of the observation structure for FHIR (RDF-star syntax [33]):

```
:x fhir:DiagnosticReport.result :ui-int_input , :ui-quant_input , … .

:ui-int_input fhir:Observation.valueInteger :v ; {| tpl:inputType xsd:integer |}
   fhir:Observation.code :c . {| tpl:hidden true |}

 :ui-quant_input fhir:Observation.code :c ; {| tpl:hidden true |}
   fhir:Observation.valueQuantity [
      fhir:Observation.Quantity.system :s ; {| tpl:hidden true |}
      fhir:Observation.Quantity.code :c ; {| tpl:hidden true |}
      fhir:Observation.Quantity.value :v . {| tpl:inputType xsd:numeric |} ] .
```

**Fig. 4.** Example FHIR observation structure[4].

In FHIR, an observation is indicated using the `fhir:DiagnosticReport.result` predicate. A concrete integer-type observation (`:ui-int_input`) is reported using the `fhir:Observation.valueInteger` predicate; we annotate this statement with its expected integer datatype (`tpl:inputType xsd:integer`). The associated terminology code is reported using `fhir:Observation.code`; the annotation indicates that the code will be hidden in a UI (`tpl:hidden true`). A quantified observation (`:ui-quant_input`) reports on quantified values, and uses the `fhir:Observation.valueQuantity` predicate to indicate a nested value composed of the coding system (`tpl:hidden`), a unit code (`tpl:hidden`), and the concrete input value (`tpl:inputType xsd:numeric`); similar annotations are provided as above. Hence, an observation structure contains all data needed for a self-contained EHR record; reporting concrete input values as well as relevant terminology and unit codes. In Section 4.1, we show how the aforementioned annotations are utilized to guide automated UI generation. We utilize the RDF-star annotation syntax [33] to conveniently annotate this code to guide the UI generation process; we use Apache Jena [34] to convert RDF-star syntax into N3 code.

### 3.3 Parametrized Semantic User Interface Code

To support a particular UI format (e.g., HTML+RDFa), our UI generation framework is loaded with snippets of *parametrized UI codes* per observation type (e.g., integer, string, boolean). The UI code is parametrized in that it includes placeholders for attributes both from the *data input needs* (Section 3.1) and *observation structure* (Section 3.2), which will be filled in during UI generation. After generation, the instantiated UI code will be used to submit a semantically annotated, self-contained health data record. As noted by Schuler et al. [10], the overall number of datatypes in openEHR is small, as is the case for FHIR datatypes, meaning that these UI code snippets can be easily

[4] Concrete objects (e.g., `_:v`, `_:c`) are not meaningful and not part of the generated UI.

developed. Below, we show parametrized HTML+RDFa and Yail code for numeric observation input (placeholders are wrapped in "_"):

```
_prefix-label_<input type='number' id='_id_' min='_min_' max='_max_' step='_step_' prop-
erty='_property_' />_suffix-label_
```

**Fig. 5.** Example parametrized UI code for HTML+RDFa.

```
{  "$Type": "HorizontalArrangement", …
   "$Components": [ {
      "$Type": "Label", "Text": "_label_"
   }, {
      "$Type": "TextBox", "NumbersOnly": "True",
      "ObjectType": "_datatype_", "PropertyURI": "_property_"
} ] }
```

**Fig. 6.** Example parametrized UI code for Yail (JSON).

The observation structure (Section 3.2) will determine the utilized `_property_` (e.g., `fhir:valueInteger`); whereas data reporting needs (Section 3.1) will dictate the label (`_label_`), and, for HTML+RDFa, the id value (`_id_`), value ranges (`_min_`, `_max_`) and required precision (`_step_`); for Yail, the expected datatype (`_datatype_`).

### 3.4 UI Templates to Guide User Interface Generation

A *UiTemplate* indicates what observation structure (Section 3.2) to utilize for a concrete data reporting need (Section 3.1), and guides the instantiation of the UI code (Section 3.3). Hence, a *UiTemplate* is specific to both an EHR standard and UI format. Below, we show an example UiTemplate for FHIR and HTML+RDFa:

```
:tpl-int_range_field a tpl:UiTemplate ;
   tpl:select :select-int_range_field
   tpl:generate [
      tpl:ui :ui-int_input ;
      tpl:placeholders ('_code_' '_prefix_' '_id_' '_min_' '_max_' '_step_' '_suffix_') ;
      tpl:values ( ?code ?label ?id ?min ?max ?step ?range ) ] .
```

**Fig. 7.** Example UiTemplate for FHIR and HTML+RDFa.

The selector (`tpl:select`) will select *certain data reporting needs*, in this case, pertaining to range-restricted integer inputs such as `:diagnose_cough_wheezing_stridor` (Fig. 1):

```
?action a fhir:ActivityDefinition ;
   fhir:ActivityDefinition.title ?label .
   fhir:ActivityDefinition.observationResultRequirement ?req .
?req fhir:ObservationDefinition.permittedDataType :dt_integer, ?range ;
   fhir:ObservationDefinition.code ?code .
?range!fhir:Range.low fhir:Quantity.value ?min .
?range!fhir:Range.high fhir:Quantity.value ?max .
(?min " - " ?max) string:concatenation ?range . # builtin for concatenating strings [19]
```

**Fig. 8**. Example *UiTemplate* selector (N3 code).

The generator (`tpl:generate`) will select the *observation structure item* to be utilized, in this case, element `:ui_int_input` (Fig. 4); and instantiate the associated *parametrized UI code*, replacing the listed placeholders with variable values from the selector.

In the following section, we detail the process that applies these *UiTemplates* to generate an output UI.

## 4 Health User Interface Generation Pipeline

This section describes the UI generation pipeline based on the input artifacts described in the prior section. Fig. 9 gives an overview of this pipeline:

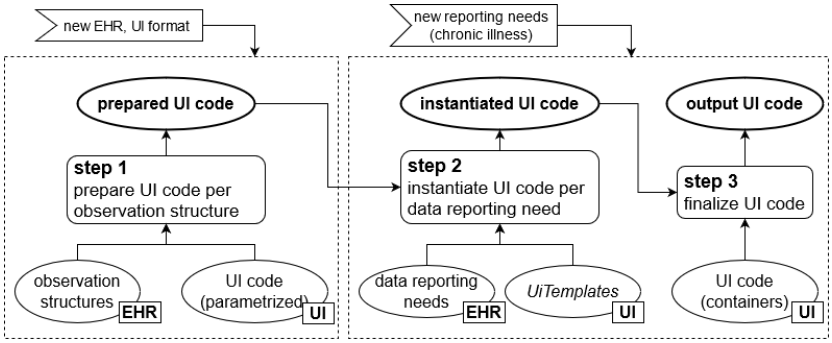


**Fig. 9.** User Interface Generation Pipeline.

Per new EHR standard and UI format, **step 1** will prepare tailored UI code from the parametrized UI snippets (e.g., Fig. 5) to report on EHR observation structures (e.g., Fig. 4)—resulting in *prepared UI code*. This step is only needed when supporting a new EHR standard or UI format (or, when using *new* UI elements to represent a datatype, such as a slider). For a given set of data reporting needs (e.g., Fig. 1), as required by a particular illness, **step 2** will fully instantiate the *prepared UI code*, guided by a set of *UiTemplates* (e.g., Fig. 7)—resulting in *instantiated UI code* per concrete reporting need. Finally, **step 3** will collect these instantiated UI codes into a single UI structure. Below, we describe each step in more detail. Our code repository includes the intermediate output for each step in the pipeline [20].

### 4.1 Step 1: Prepare UI code per Observation Structure

Each individual observation type (e.g., integer, string, boolean) is encoded using a custom UI code snippet (Section 3.3). After this step, each item in an observation structure (Section 3.2) will be represented using a group of suitable UI codes.

For instance, the `:ui-int_input` observation item (Fig. 4) includes multiple predicates that are annotated to guide UI generation (e.g., `tpl:hidden`, `tpl:inputType`). Based on these annotations, together with parametrized UI code snippets, N3 rules [20] will generate the following HTML+RDFa code:

```
<div property='fhir:DiagnosticReport.result'>
   <span style='display: none' property='fhir:Observation.code'
      resource='_code_'></span>
   <input type='number' id='_id_' min='_min_' max='_max_' step='_step_'
      property='fhir:Observation.valueInteger' />
```

```
</div>
```

**Fig. 10.** Prepared HTML+RDFa code for an observation structure item.

A hidden `<span>` element is utilized to represent the `tpl:hidden` terminology code; an `<input>` element represents the input value with `tpl:inputType xsd:integer`; and a container `<div>` element represents the entire observation. Filled-in placeholders, using the predicates from the observation structure, are underlined (e.g., `fhir:Observation.valueInteger`).

This step only needs to be performed once, per pair of EHR observation structure and UI format. As can be seen in Fig. 10, several placeholders are remaining (e.g., `_id_`, `_min_`, `_max_`), which will be instantiated in the following step.

### 4.2 Step 2: Instantiating UI codes per Data Reporting Need

A particular data reporting need (e.g., cough severity; Section 3.1) will have to be reported using a certain observation structure (Section 3.2). This step will fully instantiate the UI codes prepared in the prior step for each data reporting need. After this step, each concrete data reporting need (e.g., `:diagnose_cough_wheezing_stridor`) will be represented with a fully instantiated UI code.

The *UiTemplate* `:tpl-int_range_field` (Fig. 7) selects all data reporting needs associated with ranged-integer inputs (`:select-int_range_field`), such as `:diagnose_cough_ wheezing_stridor` (Fig. 1). For each selected reporting need, the template then copies and instantiates the UI code prepared for `:ui-int_input` (see prior step), filling in the listed placeholders using variable values returned by the selector. E.g., for `:diagnose_cough_wheezing_stridor`, the following output HTML+RDFa code is generated (placeholders filled in this step are underlined):

```
<div property='fhir:DiagnosticReport.result'>
   <span style='display: none' property='fhir:Observation.code'
      resource='copd:code_cough'></span>
   <input type='number' id='code_cough' min='1' max='10' step='1'
      property='fhir:Observation.valueInteger' />
</div>
```

**Fig. 11.** Fully instantiated HTML+RDFa code for a data reporting need.

### 4.3 Step 3: Finalize UI code Per Set of Data Reporting Needs

Finally, instantiated UI codes will be collected into a single UI structure, such as an HTML `<table>` or Yail screen (JSON object). This will represent a concrete report of observations: in openEHR, this is called a composition, in FHIR, a diagnostic report. Per UI format, an enclosing container structure is specified, and an N3 rule will collect the instantiated UI codes into this container.

## 5 Conclusion and Future Work

In this paper, we presented a UI generation framework based on (a) semantic descriptions of required health data and their constraints, (b) semantically annotated structures

for reporting observations, (c) a set of parametrized UI codes in a target UI format, and (d) a set of *UiTemplates* to guide UI generation. We gave a short overview of two open Electronic Health Record (EHR) standards, namely HL7 FHIR and openEHR, and showed how they can be utilized to encode (a) and (b). Further, we showed how semantic UI formats, i.e., supporting the semantic annotation of input data (e.g., HTML+RDFa, Yail), can be instantiated into a UI that submits a self-contained EHR record and performs input validation (to the extent allowed by the UI format). The N3 code implementing our UI generation framework is online [20].

Future work involves trying out different input artefacts (e.g., openEHR archetypes and templates) and testing the robustness of our framework. Further, we aim to customize the presentation of certain datatypes, such as limited-range integers, with more suitable UI elements (e.g., sliders). We will investigate the generation of other UI formats (e.g., Android XML, XUL). To ensure a good user experience, a generated UI will have to be manually customized, since generated UIs rather straightforwardly represent the data reporting needs in a single container. E.g., when faced with large numbers of inputs, it may be needed to spread input fields over multiple screens. Also, our UI generation implementation expects a strictly hierarchical UI structuring (although this is rather standard for UIs). Regarding openEHR, N3 represents *yet another* openEHR serialization on top of AOM, XML and JSON. Moreover, the N3 serialization is rather ad-hoc and not a proper semantic representation: e.g., ADL2OWL [23] converts cardinality and range constraints into OWL restrictions. Another approach could be to representing ADL constraints using SHACL [35].

**Acknowledgements.** Funding for this research is provided by University of New Brunswick Research Fund Competition 2020 (RF Explore).

## References

1. Packer, T.L., Boldy, D., Ghahari, S., Melling, L., Parsons, R., Osborne, R.H.: Self-management programs conducted within a practice setting: who participates, who benefits and what can be learned? Patient Educ. Couns. 87, 93–100 (2012). https://doi.org/10.1016/j.pec.2011.09.007.
2. Van Woensel, W., Baig, W.H., Abidi, S.S.R., Abidi, S.R.: A Semantic Web Framework for Behavioral User Modeling and Action Planning for Personalized Behavior Modification. In: 10th International Conference on Semantic Web Applications and Tools for Life Sciences. CEUR, Rome, Italy (2017).
3. Rose-Davis, B., Van Woensel, W., Stringer, E., Abidi, S.R., Abidi, S.S.R.: Using Artificial Intelligence-Based Argument Theory To Generate Automated Patient Education Dialogues For Families Of Children With Juvenile Idiopathic Arthritis. In: 17th World Congress on Medical and Health Informatics (MEDINFO'19), Aug 26-30. , Lyon, France (2019).
4. Johnston, N.W., Lambert, K., Hussack, P., Al, E.: Detection of COPD Exacerbations and compliance with patient-reported daily symptom diaries using a smart phone-based information system [corrected]. Chest. 144, 507–514 (2013).

https://doi.org/10.1378/chest.12-2308.
5. Van Woensel, W., Roy, P.C., Abidi, S.R., Abidi, S.S.R.: A Mobile and Intelligent Patient Diary for Chronic Disease Self-Management. In: Studies in Health Technology and Informatics (2015). https://doi.org/10.3233/978-1-61499-564-7-118.
6. da Luz Diaz, R., de Oliveira Lima, M., Alves, J.G.B., Van Woensel, W., Naqvi, A., Take, Z., Abidi, S.S.R.: A Digital Health Platform to Deliver Tailored Early Stimulation Programs for Children With Developmental Delay. In: 17th World Congress on Medical and Health Informatics (MEDINFO'19), Aug 26-30. , Lyon, France (2019).
7. Apache Foundation: Cordova, https://cordova.apache.org/, last accessed 2021/07/15.
8. HL7 International: HL7 Fast Health Interop Resources (FHIR), https://www.hl7.org/index.cfm.
9. openEHR Foundation: openEHR, https://specifications.openehr.org/.
10. Schuler, T., Garde, S., Heard, S., Beale, T.: Towards automatically generating graphical user interfaces from openEHR archetypes. Stud. Health Technol. Inform. 124, 221–226 (2006).
11. openEHR Foundation: Guideline Definition Language (GDL), https://specifications.openehr.org/releases/CDS/latest/GDL.html.
12. HL7 International: Clinical Quality Language (CQL), https://cql.hl7.org/.
13. Sutton, D.R., Fox, J.: The syntax and semantics of the PROforma guideline modeling language. J. Am. Med. Inform. Assoc. 10, 433–43 (2003). https://doi.org/10.1197/jamia.M1264.
14. Riano, D.: The SDA Model: A Set Theory Approach. In: Twentieth IEEE International Symposium on Computer-Based Medical Systems (CBMS'07). pp. 563–568. IEEE (2007). https://doi.org/10.1109/CBMS.2007.110.
15. Brush, J.E., Radford, M.J., Krumholz, H.M.: Integrating Clinical Practice Guidelines Into the Routine of Everyday Practice. Crit. Pathways Cardiol. A J. Evidence-Based Med. 4, 161–167 (2005). https://doi.org/10.1097/01.hpc.0000173342.41305.b0.
16. Wusteman, J.: About XML: from Ghostbusters to libraries – the power of XUL. Libr. Hi Tech. 23, 118–129 (2005). https://doi.org/10.1108/07378830510586757.
17. Patton, E.W., Tissenbaum, M., Harunani, F.: MIT App Inventor: Objectives, Design, and Development. In: Kong, S.-C. and Abelson, H. (eds.) Computational Thinking Education. pp. 31–49. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-6528-7_3.
18. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax, https://www.w3.org/TeamSubmission/n3/, last accessed 2021/07/15.
19. Arndt, D., Van Woensel, W., Tomaszuk, D.: Notation3: Draft Community Group Report, https://w3c.github.io/N3/spec/, last accessed 2021/07/15.
20. Van Woensel, W.: UI generation from EHR data descriptions, https://github.com/william-vw/ui_gen.
21. Beale, T.: Archetypes: Constraint-based Domain Models for Future-proof Information Systems. In: Eleventh OOPSLA Workshop on Behavioral Semantics. Serving the Customer (2002).

22. Fernández-Breis, J.T., Maldonado, J.A., Marcos, M., Legaz-García, M. del C., Moner, D., Torres-Sospedra, J., Esteban-Gil, A., Martínez-Salvador, B., Robles, M.: Leveraging electronic healthcare record standards and semantic web technologies for the identification of patient cohorts. J. Am. Med. Inform. Assoc. 20, e288-96 (2013). https://doi.org/10.1136/amiajnl-2013-001923.
23. Lezcano, L.: On the Integration of Clinical Archetypes with Ontologies and Rules. Presented at the (2013). https://doi.org/10.4018/978-1-4666-3000-0.ch005.
24. Ramesh, S.: Medblocks UI, https://medblocks-ui.vercel.app/?path=/story/introduction--page, last accessed 2021/07/15.
25. Mozilla: Web Components, https://developer.mozilla.org/en-US/docs/Web/Web_Components, last accessed 2021/07/15.
26. Pazos, P.: openEHR Discussion Forums: post, https://discourse.openehr.org/t/will-an-openehr-implementation-provide-ui-to-create-and-render-records-based-on-a-template/799/3, last accessed 2021/07/15.
27. Berners-Lee, T.: Notation 3 Logic, https://www.w3.org/DesignIssues/Notation3, last accessed 2021/07/15.
28. Arndt, D., Schrijvers, T., De Roo, J., Verborgh, R.: Implicit quantification made explicit: How to interpret blank nodes and universal variables in Notation3 Logic. J. Web Semant. 58, 100501 (2019). https://doi.org/10.1016/J.WEBSEM.2019.04.001.
29. Word Wide Web Consortium: W3C Notation 3 Community Group, https://www.w3.org/community/n3-dev/.
30. HL7 International: Resource Description Framework (RDF) Representation, https://www.hl7.org/fhir/rdf.html.
31. Arndt, D., Woensel, W. Van: Towards Supporting Multiple Semantics of Named Graphs Using N3 Rules. In: Proceedings of the 13th RuleML+RR 2019 Doctoral Consortium and Rule Challenge, September 16-19, 2019 - Bolzano, Italy, September 16-24, 2019. CEUR-WS.org (2019).
32. SNOMED CT, http://www.snomed.org/.
33. Hartig, O., Champin, P.-A., Kellogg, G., Seaborne, A.: RDF-star and SPARQL-star: Draft Community Group Report, https://w3c.github.io/rdf-star/cg-spec/editors_draft.html, last accessed 2021/07/16.
34. Apache: Apache Jena, https://jena.apache.org/.
35. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL): W3C Recommendation, https://www.w3.org/TR/shacl/, last accessed 2021/07/16.