User Interface Templates

for the openEHR specifications stack

The UITemplate Model Specification

v0.3.1

Pablo Pazos Gutierrez
pablo.pazos@cabolabs.com

Version	Author	Date	Changes
v0.3	Pablo Pazos	20180312	Initial spec, review from original work: https://www.slideshare.net/pablitox/generacin-automtica-de-interfaces-de-usuario-para-sistemas-de-informacin-clnicos-basados-en-una-metodologa-multinivel
v0.3.1	Pablo Pazos	20180313	Added information about the UI generation modes on the UI Generation Framework section
v0.3.2	Pablo Pazos	20200915	Added extra fields (label, bindings, language) and JSON instance sample.

Introduction

The current openEHR specifications don't include artifacts to model, specify, process and generate user interfaces for front-end applications. We believe this specification is needed to standardize the criteria on clinical user interfaces, improve usability, leverage the clinical modeling process and reuse current archetypes and templates. By defining a new layer of specifications, we can add UI directives over templates, without the need of changing the information model or add extra non-standard metadata to archetypes and templates.

Design criteria

We believe for a good and low coupled design, UI directives or UI metadata should not be included in the current models, but in a new one that references the current models.

UI directives include information about the position, size, order, grouping, layout and style of UI elements or "Controls" that are used for data input or output (display). This initial specification doesn't include the definition of rules (show/hide under conditions, workflow control, highlight under conditions, etc).

Each "Control" has a reference to a node from an Operational Template (OPT), and the type of Control will depend on the DataType of the node in the OPT. First we'll focus on simple Controls for single data input or output. After reviewing the current version of this specification and gathering more requirements, we can start adding more complex Controls to the specification, or even define those as extensions or "plugins".

We defined a small terminology to define the different types of controls. Each control type matches with an openEHR datatype. The idea is to expand this terminology after receiving input from the community.

The UITemplate model will have references only to OPTs. This will lead to a layered design, with low coupling and better maintainability. Of course, OPTs include information from archetypes, and those include information from the information model.

Controls are organized in Views (that represent a form or a screen), and each View complies with a Layout (general visual organization). Having a huge OPT doesn't mean that all the Controls for it should be displayed on the same screen, several Views can be defined for one OPT to display parts of the nodes on each View. This allows to create more usable and user-friendly screens.

The specification has three parts, the first part is the UITemplate Model, the second part is the UITemplate XML Format (with a correspondent XML Schema), and the third is the UI Generation Framework.

The UITemplate Model

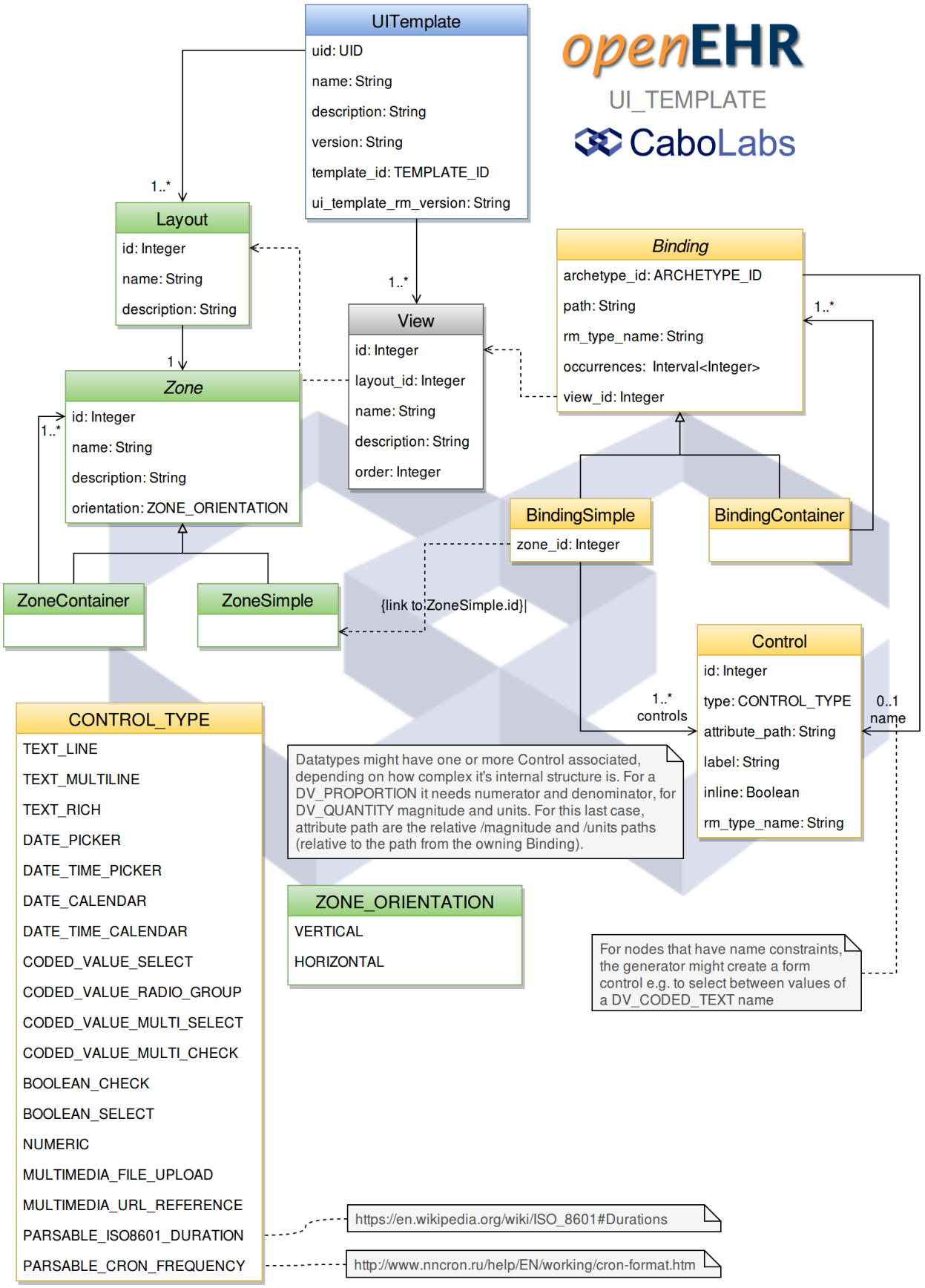

Fig. 1: UITemplate Reference Model

uitemplate.UITemplate

Definition of a UITemplate, contains layouts, views, bindings and controls.

Attribute	Description	Notes
uid: UID	Unique identifier for this UITemplate, can be UUID or OID.	We can think of a similar format to the archetype ID that includes a version axis.
name: String	Name of this UITemplate	TODO: name should be I18N like in archetypes, so this should really be a code like at0000. Another solution is to think of this as context-specific and have the name in the same language as the referenced OPT.
description: String	Description of this UITemplate	Idem.
version: String	Version of this UITemplate following the semver format.	
teplate_id: TEMPLATE_ID	Identifier of the OPT that this UITemplate is using.	
ui_template_rm_version: String	Version of the UITemplate spec that this UITemplate is compliant with.	
layouts: Set<Layout>	Layouts defined for this UITemplate. Should be at least one Layout.	
views: Set<View>	Views defined for this UITemplate. Should be at least one View.	
language: String	Two digit language code, based on the OPT language.	All the free texts inside the UITemplate will be in this language.
bindings: Set<Binding>	Bindings to OPT nodes.	

uitemplate.layout.Layout

General organization of Controls on a View.

Attribute	Description	Notes
id: Integer	Used to identify Layouts in the UITemplate and	

	define references from Views.	
name: String		TODO: name should be I18N like in archetypes, so this should really be a code like atNNNN. Another solution is to think this as context-specific and have the name on the same language as the referenced OPT.
description: String		Item
zone: Zone	Root Zone of the Layout.	

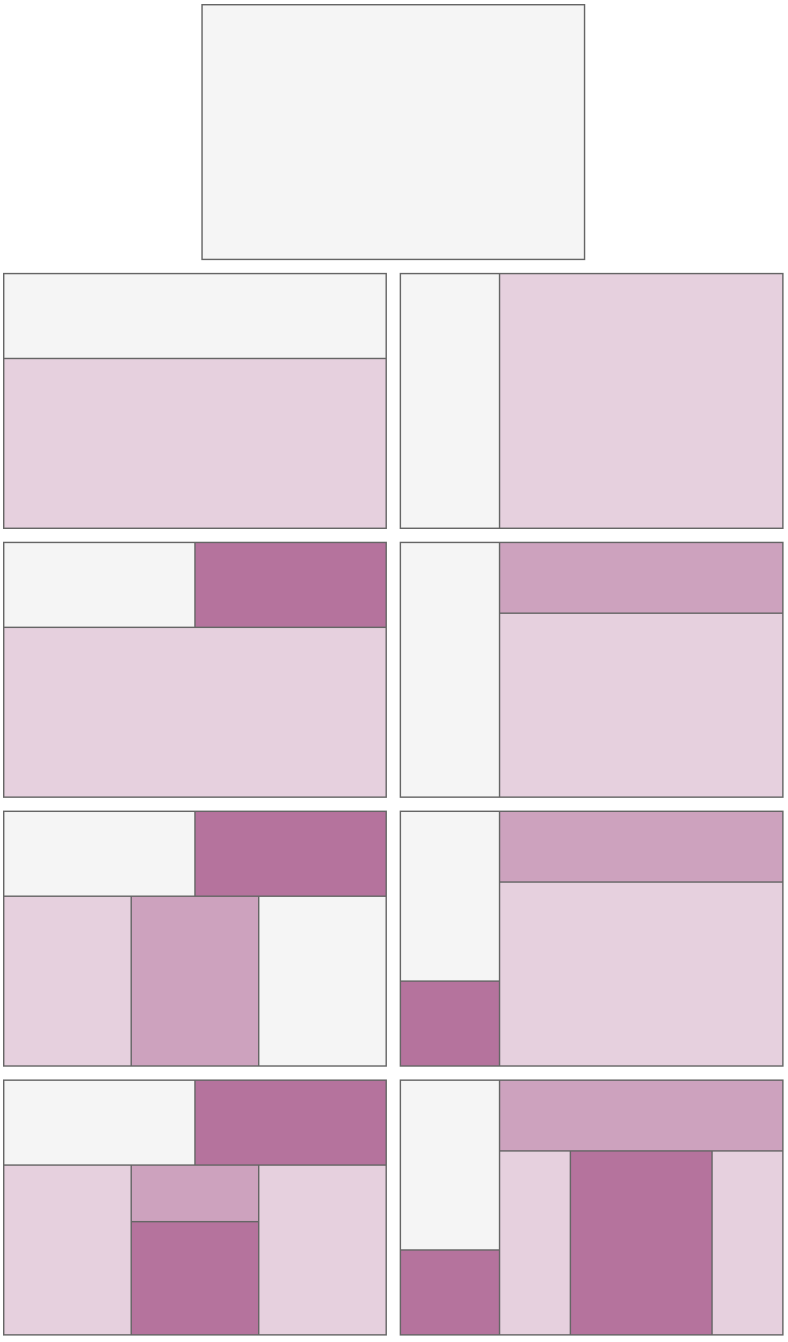

Fig. 2: Layout organization in Zones, left column starts with horizontal Zones, right column starts with vertical Zones

uitemplate.layout.Zone

Area from a Layout where other zones are defined or controls should be displayed. Is an organizational element.

Attribute	Description	Notes
id: Integer		
name: String		TODO: name should be I18N like in archetypes, so this should really be a code like atNNNN. Another solution is to think of this as context-specific and have the name in the same language as the referenced OPT.
description: String		Item
orientation: ZONE_ORIENTATION	The zone can be horizontal or vertical, this defines how other Zones or Controls should be displayed in this Zone.	The orientation will always alternate between H and V, so only the first orientation is really required, the rest could be derived from the position of the Zone in the hierarchy formed by parent-child Zones.

uitemplate.layout.ZoneContainer inherits from Zone

Defines a group of Zones, following an orientation.

Attribute	Description	Notes
zones: List<Zone>	Zones contained on this ZoneContainer.	

uitemplate.layout.ZoneSimple inherits from Zone

Defines a Zone where Controls should be displayed.

Attribute	Description	Notes

uitemplate.layout.ZONE_ORIENTATION <Enum>

Defines the valid values for the Zone.orientation attribute.

Value	Description	Notes
VERTICAL	The Zone should organize it's contents in a vertical way, one on top of the following.	
HORIZONTAL	The Zone should organize it's contents in an horizontal way, one at the left of the following.	

uitemplate.view.View

Defines one single screen or form to be displayed following a Layout.

Attribute	Description	Notes
id: Integer	Identifier of this View inside the UITemplate.	
layout_id: Integer	Reference to a Layout.	The reference to the Layout in the UML model can be specified without this attribute. This attribute is needed by the serialization format to avoid reference loops.
name: String		TODO: name should be I18N like in archetypes, so this should really be a code like atNNNN. Another solution is to think this as context-specific and have the name on the same language as the referenced OPT.
description: String		Item
order: Integer	Relative order of the Views inside the UITemplate. Allows to create menus using the order to sort the elements to display. Allows that order to be independent of the order the Views are declared in the UITemplate. Also the order can be overwritten by software applications, adjusting the items on the menus to specific usage contexts.	

uitemplate.binding.Binding

Allows to specify references from the UITemplate model to the Archetype Model, to bind UI Controls with archetype nodes. The valid archetypes are the ones defined inside the referenced OPT, and the paths should also exist on the OPT.

Attribute	Description	Notes

archetype_id: ARCHETYPE_ID	Identifier of the referenced archetype, existing on the referenced OPT.	
path: String	Valid path inside the archetype with id = archetype_id.	
rm_type_name: String	Type referenced by the path.	Need to discuss how we will deal with alternatives (same path, multiple data types). One way is to only allow one data type per binding. If two Controls for the same path and different data types should be on a View, two Bindings should be created.
occurrences: Interval<Integer>	Occurrences of this binding in the UI. Allows multiple data entries for the same OPT node, if the node has multiple occurrences in the data structure definition. This attribute can narrow the possibilities but not break the current constraints for the node.	
view_id: Integer	Reference to a View.	The reference to the View in the UML model can be specified without this attribute. This attribute is needed by the serialization format to avoid reference loops.
name: Control	Control used to display an input for a node name when the name of a node (LOCATABLE) is constrained as a DV_CODED_TEXT in the OPT.	

uitemplate.binding.BindingSimple inherits from Binding

Defines a Control container.

Attribute	**Description**	**Notes**
zone_id: Integer	Identifier of the Zone that this group of Controls should be displayed in.	The reference to the Zone in the UML model can be specified without this attribute. This attribute is needed by the serialization format to avoid reference loops.
controls: List<Control>	Controls contained on this Binding.	
label: String	Text label associated with the fields in this binding.	By default will be a copy of the text associated with the path from the template, but could be overridden by a specified value.

uitemplate.binding.BindingContainer inherits from Binding

Defines a container of Bindings, allowing to represent the OPT tree and contain groups of Controls (BindingSImple).

Attribute	Description	Notes
bindings: Set<Binding>		

uitemplate.binding.control.Control

Represents a UI element.

Attribute	Description	Notes
id: Integer		
type: CONTROL_TYPE	The type defines what to display on a UI. The Control type depends on the corresponding openEHR data type referenced by a Binding.rm_type_name.	For now only simple types are defined, this model can be extended to add more types and complex types.
attribute_path: String	If the parent Binding path is to a DV_QUANTITY, the Control.attribute_path allows to reference attributes inside the DV_QUANTITY like /units and /magnitude, so it is a relative path to the Binding.path. This allows easy data binding from data input to the openEHR IM.	
label: String	Optional label to be displayed near this Control. By default the label is the node text from the OPT.	This is useful for Controls associated with structured data values, to know to which attribute each Control is associated with. For instance, a DV_QUANTITY will have a Control for 'magnitude' and another for 'units'.
inline: Boolean	If true, indicates that the Control on the UI should be displayed on the same line as the previous Control in the same Binding.	
rm_type_name: String	The specific openEHR RM type that corresponds to this Control, like String for DV_QUANTITY.units	

uitemplate.binding.control.CONTROL_TYPE <Enum>

Defines the valid values for the Control.type attribute.

Value	Description	Notes
TEXT_LINE	Single line text input or display.	
TEXT_MULTILINE	Multiline text input or display.	
TEXT_RICH	Rich/styled text editor i.e. a WYSIWYG editor.	
DATE_PICKER	Simple date selector.	
DATE_TIME_PICKER	Simple date and time selector.	
DATE_CALENDAR	Date selector from a calendar.	
DATE_TIME_CALENDAR	Date and time selector from a calendar.	
CODED_VALUE_SELECT	List of items with codes, just one can be selected at a time.	
CODED_VALUE_RADIO_GROUP	List of items with codes displayed as a radio button group, only one can be selected at a time.	
CODED_VALUE_MULTI_SELECT	List of items with codes, more than one can be selected at a time.	
CODED_VALUE_MULTI_CHECK	List of items with codes displayed as a check button list, more than one can be selected at a time.	
BOOLEAN_CHECK	Boolean input or display using a check box.	
BOOLEAN_SELECT	Boolean input using a list of values from where only one can be selected at a time.	
BOOLEAN_RADIO_GROUP	Boolean input or display using a radio button group, only one can be selected at a time.	
NUMERIC	Input or display of a numeric value.	
MULTIMEDIA_FILE_UPLOAD	Input field to upload a multimedia file.	
MULTIMEDIA_URL_REFERENCE	Input field that allows to reference an external multimedia resource. This can be a simple text input where an URL is pasted, or an integration with an external system to look for the desired resource and grabs the URL internally, for instance an image from a PACS.	
PARSABLE_ISO8601_DURATION	Input or display control for duration expressions in ISO8601. Can be a complex UI element that allows the user to create the duration expressions in a friendly way or a simple text input where the user needs to paste the expression.	https://en.wikipedia.org/wiki/ISO_8601#Durations
PARSABLE_CRON_FREQUENCY	Input or display control for frequency expressions in cron format. Can be a complex UI element that allows the user to create the frequency	http://www.nncron.ru/help/EN/working/cron-format.htm

	expressions in a friendly way or a simple text input where the user needs to paste the expression.	
...		

UITemplate XML Format

The processable expression for UITemplate instances will be defined in a XSD. This is work in progress and will be published soon.

UITemplate XML Format Sample

TBD

UITemplate JSON Format

The processable expression for UITemplate instances will be defined in a JSON Schema. This is work in progress and will be published soon.

UITemplate JSON Format Sample

This instance is based on the physical_activity_document.en.v1 Operational Template published here https://github.com/ppazos/testehr/blob/master/src/main/resources/opts/Physical_Activity_Document.opt


```
{
 "_type": "UITemplate",
 "uid": "132456",
 "name": "Physical Activity",
 "description": "Physical Activity Document",
 "language": "en",
 "version": "1.0.0",
 "template_id": "physical_activity_document.en.v1",
 "ui_template_rm_version": "0.1",
 "layouts": [
   {
     "_type": "Layout",
     "id": 1,
     "name": "main",
     "description": "",
     "zone": {
       "_type": "ZoneContainer",
       "id": 1,
       "name": "root zone",
       "description": "root zone of the layout",
       "orientation": "HORIZONTAL",
       "zones": [
         {
           "_type": "ZoneSimple",
           "id": 2,
```

```
          "name": "left col",
          "description": "",
          "orientation": "VERTICAL"
        },
        {
          "_type": "ZoneSimple",
          "id": 3,
          "name": "right col",
          "description": "",
          "orientation": "VERTICAL"
        }
      ]
    }
  }
],
"views": [
  {
    "_type": "View",
    "id": 1,
    "name": "main view",
    "description": "",
    "order": 0,
    "layout_id": 1
  }
],
"bindings": [
  {
    "_type": "BindingSimple",
    "label": "Type of activity",
    "archetype_id": "openEHR-EHR-OBSERVATION.physical_activity.v1",
    "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value",
    "rm_type_name": "DV_CODED_TEXT",
    "occurrences": {
      "low": 1,
      "high": 1
    },
    "view_id": 1,
    "zone_id": 1,
    "controls": [
      {
        "_type": "Control",
        "id": 1,
        "type": "CODED_VALUE_RADIO_GROUP",
        "attribute_path": "/defining_code/code_string",
        "inline": true,
        "rm_type_name": "String"
      }
    ]
  },
  {
```

```
    "_type": "BindingSimple",
    "label": "Time of activity",
    "archetype_id": "openEHR-EHR-OBSERVATION.physical_activity.v1",
    "path": "/data[at0001]/events[at0002]/time",
    "rm_type_name": "DV_DATE_TIME",
    "occurrences": {
      "low": 1,
      "high": 1
    },
    "view_id": 1,
    "zone_id": 1,
    "controls": [
      {
        "_type": "Control",
        "id": 2,
        "type": "DATE_TIME_CALENDAR",
        "attribute_path": "/value",
        "inline": true,
        "rm_type_name": "DateTime"
      }
    ]
  },
  {
    "_type": "BindingSimple",
    "label": "Duration",
    "archetype_id": "openEHR-EHR-OBSERVATION.physical_activity.v1",
    "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0009]/value",
    "rm_type_name": "DV_QUANTITY",
    "occurrences": {
      "low": 1,
      "high": 1
    },
    "view_id": 1,
    "zone_id": 2,
    "controls": [
      {
        "_type": "Control",
        "id": 3,
        "type": "NUMERIC",
        "attribute_path": "/magnitude",
        "inline": true,
        "rm_type_name": "Real"
      },
      {
        "_type": "Control",
        "id": 4,
        "type": "CODED_VALUE_RADIO_GROUP",
        "attribute_path": "/units",
        "inline": true,
        "rm_type_name": "String"
```

```
      }
    ]
  },
  {
    "_type": "BindingSimple",
    "label": "Intensity",
    "archetype_id": "openEHR-EHR-OBSERVATION.physical_activity.v1",
    "path": "/data[at0001]/events[at0002]/data[at0003]/items[at0010]/value",
    "rm_type_name": "DV_CODED_TEXT",
    "occurrences": {
      "low": 1,
      "high": 1
    },
    "view_id": 1,
    "zone_id": 2,
    "controls": [
      {
        "_type": "Control",
        "id": 1,
        "type": "CODED_VALUE_RADIO_GROUP",
        "attribute_path": "/defining_code/code_string",
        "inline": true,
        "rm_type_name": "String"
      }
    ]
  }
]
}
```


UI Generation Framework

The UI Generation Framework defines the steps/phases of a generic UI generator based on declarative UI definitions.

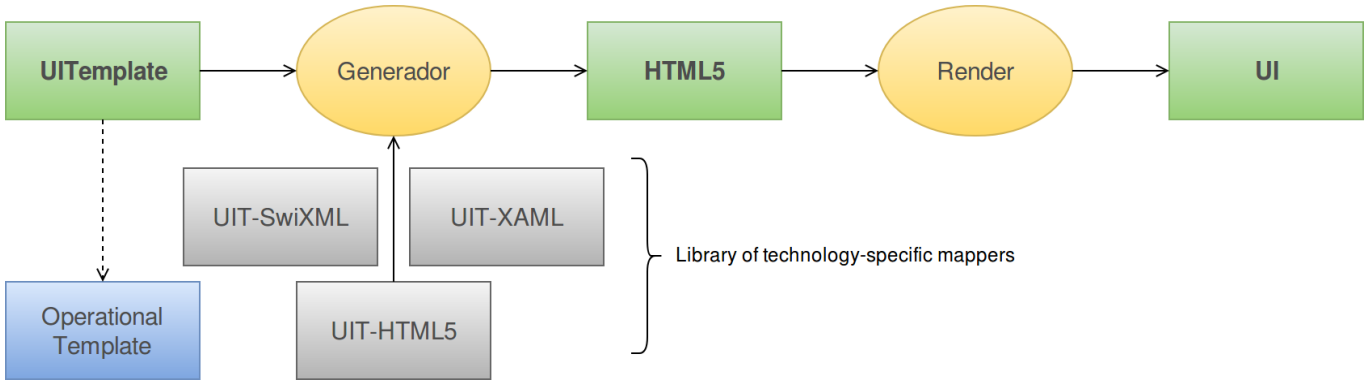

Fig. 3: Declarative UI Generation Framework phases

The diagram in fig. 3 shows in green the UI artifacts, in yellow the software components that process and generates artifacts, in gray the configurations for the UI generator that contain mappings from the UITemplate to specific technologies, and in blue is the referenced OPT from the UITemplate.

Our goal is to define a pluggable architecture for the generic UI Generator, so mappings to new technologies can be defined without the need of modifying the generator. If all the technology-specific declarative UI definitions are expressed in XML (as most are), the mappers can be represented by XSLTs or similar, mapping from the UITemplate XML Format, to XAML, SwiXML, etc. But XSLT can be difficult to define and manage for complex transformation, in that context specific tools might be useful, like Smooks.

We implemented a simple proof of concept generator that proved our framework design was correct. Currently we need to finish the UITemplate XML Format and build the generator based on decisions about the mapping implementation. So we need to test the pros/cons of XSLT vs. Smooks or similar vs. custom DSL (define our own mapping language).

The output of the generator are declarative definitions of UIs in three "modes": create, show and edit. The "create" is a definition that should be used for data input. The "show" is a definition for data visualization (read only). And "edit" is a definition that should display data in order to be modified. These modes are oriented to data input applications and more modes should be defined, for instance an "aggregated" mode can be used to display many data points, on the same screen, maybe from clinical documents defined by different OPTs. Or a "population" mode can be used to display information from many different patients on the same screen. We need to explore more use cases to define the extra modes.

References:

Declarative UI:
https://mycourses.aalto.fi/pluginfile.php/394893/mod_resource/content/2/declarative_user_interfaces_.pdf

Declarative UI .NET XAML:
https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf

Declarative UI iOS: https://github.com/schibsted/layout

Declarative UI Java Swing: https://github.com/swixml/Two

Declarative UI Android: https://developer.android.com/guide/topics/ui/declaring-layout.html

Declarative UI Java XAML: http://www.soyatec.com/eface/