Collective Wisdom Driving Public Health Policies

D3.3 Health Record Structure: Software prototype v1

Project Deliverable

This project has received funding from the European Union's Horizon 2020 Programme (H2020-SC1-2016-CNECT) under Grant Agreement No. 727560

D3.3 Health Record Structure: Software prototype v1

Work Package:		WP3
Due Date:		31/12/2017
Submission Date:		19/01/2018
Start Date of Project:		01/03/2017
Duration of Project:		36 Months
Partner Responsible of Deliverable:		ENG
Version:		1.1
Status:	☒ Final ☐ Draft ☐ Ready for internal Review ☒ Task Leader Accepted ☒ WP leader accepted ☒ Project Coordinator accepted	
Author name(s):	Francesco Torelli, Antonio De Nigro, Domenico Martino (ENG), Maroje Sorić, Bojan Leskošek (ULJ), Jan Janssen, Serge Autexier (DFKI), Santiago Aso (ATOS), Thanos Kiourtis (UPRC)	
Reviewer(s):	Andreas Menychtas(BIO)	Sokratis Nifakos (KI)
Nature:	☐ R – Report ☒ D – Demonstrator	
Dissemination level:	☒ PU – Public ☐ CO – Confidential ☐ RE – Restricted	

REVISION HISTORY			
Version	Date	Author(s)	Changes made
0.1	28/11/2017	ENG	First draft – Index, Holistic Health Record Manager, HHR mapping syntax
0.2	19/12/2017	ENG, ULJ, DFKI, ATOS, UPRC	Updated index, updated Holistic Health Record Manager section, renamed section from “HHR mapping syntax” to “HHR to FHIR mapping”, updated HHR to FHIR mapping section, edited HHR to FHIR mapping example, executive summary, source code section, references
1.0	31/12/2017	ENG	Fixed internal review remarks.
1.1	19/01/2018	ATOS	Quality Review. Submission to EC.

List of acronyms

FHIR	Fast Healthcare Interoperability Resource Specification
HHR	Holistic Health Record
UML	Unified Modeling Language
XML	Extensible Markup Language

Contents

List of figures

1. Executive Summary

This document describes the first implementation of the Holistic Health Record (HHR) Model and related specifications. The implementation consists of two components: a Java library and a machine-interpretable mapping from the HHR model to the FHIR[1] model.

The Java library, called HHR Manager, allows to instantiate and modify in-memory Java objects that are compliant to the HHR conceptual model (see deliverable D3.1). In order to produce it, the HHR model has been first formalized using a language called "HHR mapping language". This is an XML language, specifically designed for the HHR model, that allows to specify in a machine-interpretable way the structure of HHR types and map them to the structure of corresponding FHIR resources. The *HHR mapping language* is basically a declarative language for defining and mapping document oriented (i.e. tree-like) data structures and exploits the *FHIRPath language*[2] to navigate such structures. The HHR mapping language can be considered an alternative to the *FHIR mapping language*[3], that is currently being specified as part of the FHIR standard. The FHIR mapping language is an imperative language and arguably more powerful than the "HHR mapping language", but often produces complex descriptions. Instead the "HHR mapping language" is intended to be more lightweight.

The machine-interpretable mapping document, expressed with the "HHR mapping language", represents a formalization of the HHR model that deliverable D3.1 presents in an informal way.

The code of the HHR Manager has been partially produced in an automatic way, by exploiting the formal specification of the HHR model, and the rest has been produce in a manual way. The same formal specification will be used at runtime by the "DataConverter" component to convert HHR objects to FHIR resources (see deliverable D3.9).

In summary, the following process has been followed: (1) the "HHR mapping language" has been specified; (2) the formal specification of the HHR model has been authored using this language; (3) the HHR Manager has been implemented; (4) the formal specification has been completed by adding the mapping to FHIR.

The rest of this document is organized in two chapters. The first chapter describes the HHR Manager and the HHR mapping language. The second chapter describes how to download and use the HHR Manager and the machine-interpretable mapping to FHIR.

[1] https://www.hl7.org/fhir/
[2] http://hl7.org/fhirpath/
[3] https://www.hl7.org/fhir/mapping-language.html

2. Prototype overview

2.1. Holistic Health Record Manager

In this section we describe the Holistic Health Record Manager (HHR Manager) and its functions. The HHR Manager is a Java library that implements the HHR model. In the CrowdHEALTH platform, the HHR Manager component may be used by the Data Converter component, as shown in Figure 1.

Figure 1 Usage of *HHR Manager* in the CrowdHEALTH platform

The library contains two packages: *eu.crowdhealth.hhr.model* and *eu.crowdhealth.hhr.impl* (simply called *model* and *impl* in the following). The first package defines the HHR Manager API (IHHRManager in the above picture), i.e. a set of public Java interfaces and enumerations corresponding to the conceptual types defined by the HHR conceptual model, as reported in the deliverable D3.1. The second package implements the API.

Each leaf class of the HHR conceptual model having the stereotype <enum> corresponds to a homonymous Java enumeration defined in the *model* package. Some enumeration inherits from the parametric interface HHRType<T>. Any instance of HHRType<T> represents the reification of a subclass of T. For instance, the enumeration *ProcedureType* implements HHRType<Procedure>, i.e. each instance of *ProcedureType* represents a different type of *Procedure*.

Any non-leaf class (including classes with whatever stereotype) of the HHR conceptual model corresponds to a homonymous Java interface within the *model* package. All interfaces inherit from the top interface called HHR.

The *impl* package provides only one public class, named “HHRFactory”, that allows the creation of Java objects implementing the *model* interfaces (any other implementation class is private to the package and cannot be directly instantiated).

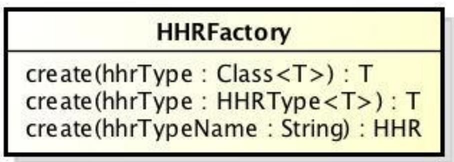

Figure 2 Class *HHRFactory*

The class HHRFactory offers three “create” methods, that return a new instance of the HHR type specified as input. The type to instantiate may be specified by passing to the *create* method the descriptor of a *model* interface (e.g. *Patient.class*), or by passing an HHRType value (e.g. *ClinicalFinding.ANEMIA*) or by passing directly the name (as *String*) of a conceptual type (e.g. “Patient” or “ANEMIA”). The first two methods are generic methods which return type corresponds exactly to the type specified by the input, while the third method has the interface HHR as return type. Therefore, while the first method allows to write safe code and is useful when the type to instantiate is statically known or determined by generic methods, the second method has to be used when no information on the type to instantiate is statically known.

HHR conceptual classes having the special attribute called “type” can be only instantiated using the create method accepting an HHRType as input. Differently by other attributes, the value of the “type” attribute cannot be changed after the creation of the instance, because it is not allowed to change the type of any created object, regardless if such a type is expressed just by the interface of the object or by the “type” attribute. An exception is raised if a client tries to instantiate an object having the “type” attribute using an interface instead than an HHRType.

Following the JavaBeans pattern, each attribute of a HHR conceptual type is represented as a Java “property”, i.e. a couple of “set and get methods. While in the HHR conceptual model all attributes have a singular name, when an attribute has a maximum cardinality greater than one the corresponding Java properties have a plural name and the corresponding methods has a collection of entities as input/output return types. Such properties are called multiple values properties. Note that the input/output collection of the get/set methods is not the value of the property but has the meaning of assigning/returning multiple values to a property; if you use a collection to set multiple values to a property and afterwards you add a new value to that collection, the values of the property will be not affected: the only way to change the values of a property is to invoke again the corresponding set method. A set method of a property *P* (i.e. a method having name “set*P*”) always substitutes all the values of the property *P*. The get method of a multiple-values property *P* (i.e. a method having name “get*P*”) never returns *null* values, but only filled or empty collections; trying to set a multiple-values property with a *null* value will raise an exception.

When an instance of an interface corresponding to an HHR conceptual class having stereotype <<role>> is created (e.g. *Patient*, that is a role of a *Person*), the resulting Java object implements both the instantiated interface (e.g. *Patient*) and the corresponding player interface (e.g. *Person*). This allows to assign an object having a certain role type also to the properties that expect the player type (but not vice versa). The implementation of each property inherited by the player interface is implemented by delegation to the player object. Therefore, setting the age of a *Person* instance is equivalent to set the age of any of its role instances (and vice versa), and all role properties return the same values of the corresponding properties of the player instance (and vice versa). In other terms, instances of different roles, having a same entity as a player, represent different views of that entity. Each view of a same entity has a distinct identity (i.e. the same *Person* may play different *Patients* that share the same properties of the played *Person*, but have also additional properties).

As explained in deliverable D3.1, all non-leaf classes of the HHR conceptual model are ontological *abstract* classes (i.e. their instances are all and only the instances of their subclasses). Correspondingly, only leaf Java interfaces can be "instantiated” (i.e. the HHRFactory will raise an exception if asked to create an instance of a non-leaf interface) [4].

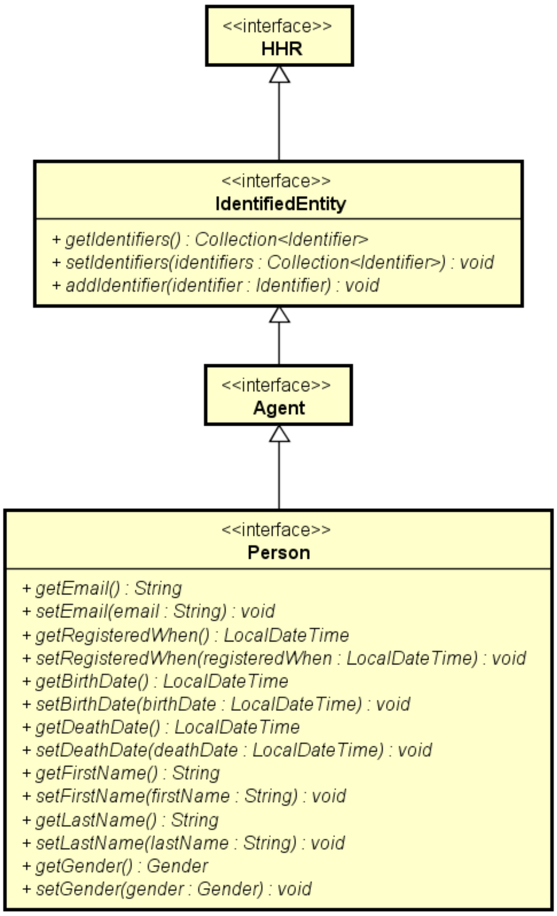

Figure 3 Specification of class *Person*

[4] Note that this is not a direct consequence of the fact that non-leaf HHR conceptual classes are ontologically abstract, but it is a design choice made to simplify the usage of the HHR API. Other implementations of the HHR model could allow to instantiate non-leaf conceptual classes, to represent entities which most specific type is yet unknown. Such a kind of implementation could allow to dynamically set a more specific type to an object, after its instantiation, and/or could allow the management of distinct Java objects that represent the very same real world entity at different levels of detail. Such advanced functionalities are currently not required by use cases; therefore, the corresponding complexity is avoided.

For instance (see section 4.3 of deliverable D3.1), the class *Person* (reported in Figure 3) is a leaf class that specializes the abstract class *Agent*. The HHR Manager allows to create an instance of *Person*, while *Agent*, *IdentifiedEntity*, HHR are not instantiable. The HHR conceptual model also contains abstract classes that are not leaf classes, e.g. although *AutomaticAgent* is a leaf class in the conceptual model, it is expected to be specialized in next versions of the HHR model and thus the corresponding Java interface is a not instantiable.

The HHR Manager assumes that no more than one one Java object is created for describing any real world entity. In any case, it is possible to create several instances that represent *generic entities*, i.e. that represent a single entity which specific identity is unknown, but which type and possible other properties are known. A generic entity is represented by an instance of a concrete subtype of IdentifiedEntity which *identifiers* are not set. For example, to represent a *Condition* identified by a *Practitioner* and reported by a *Patient* that has not specified the identity of the Practitioner, an instance of Practitioner without identifier is set as value of the attribute *performer* of the *Condition*[5].

Some Java property corresponds to UML association-ends of the HHR conceptual model. Each association of the HHR conceptual model can be navigated only in one direction, therefore only one UML association-end of each association is represented in Java.

The current implementation of the HHR model is based on a fully open word assumption, i.e. it's not possible to know if the information about an entity is complete or not. For instance, if a get method returns some values (or no value), it means that only those values (or no value) are (is) known for the corresponding property, but cannot be excluded that other values may exist for the corresponding attribute of the corresponding real world entity. The implementation does not allow to assert that no other values exist for a property. As a consequence of the open world assumption, no cardinality constraints are enforced by the implementation, with the exception of the distinction between single-value (i.e. maximum cardinality equal to 1) and multiple-value (i.e. maximum cardinality greater than 1) properties. If a property is multiple-value, then setting that property with an empty collection means just that no value of that property is known (it does not mean that no value exists). Similarly, if a property is multiple-values then setting a *null* value means that the value of that property is unknown.

2.2. HHR to FHIR mapping

The HHR model is based on the FHIR standard, although it is designed at a more conceptual level and it is more specific than the FHIR model. A mapping between the two models was firstly expressed in deliverable D3.1 in a semiformal way, and now it has been expressed in a

[5] Generic entities have been introduced after the release of the first version of the HHR model and led to remove from the model the subclasses of *Condition,* named *ConditionIdentifiedByPractitioner* and *ConditionIdentifiedByPatient*, originally introduced to distinguish the type of the *performer* of a *Condition* when the identity of the performer is unknown. Now the HHR conceptual class *Condition* is concrete.

machine-interpretable format, using an XML language, called “HHR mapping language”, specifically developed for the HHR model.

The HHR mapping language allows to express in a declarative way both the structure of the HHR model and the needed transformations to convert any HHR instance in a corresponding FHIR resource or data. The description is modularized according to the classes of the HHR model.

Each class of the HHR conceptual model corresponds to one or more FHIR resources or data type. When no explicit conversion rule is provided for an HHR type then each instance of that type will have the same representation in both the HHR model and the FHIR model. In particular no conversion is needed for primitive values and for the complex type *Identifier*.

Each tag occurrence in the XML mapping document describes a portion of the HHR model and also expresses a conversion rule for that portion of model. Each conversion rule may trigger other conversion rules (for instance, a tag <class> represents a conversion rule to convert an instance of a certain HHR type to the corresponding FHIR instance and triggers other conversion rules represented by nested tags <attribute>, to convert the attributes of the instance; each tag <attribute> on turn triggers some <class> rule to convert the values of the translated attribute).

In this document, the HHR instance to be converted to FHIR is called *source-HHR-instance*, while the FHIR resource or data record that is the result of the conversion is called *target-FHIR-instance*.

The machine-interpretable mapping can be used during a validation process to check that a *target-FHIR-instance,* already associated to a *source-HHR-instance,* has the correct type and structure or can be used during a conversion process to convert a *source-HHR-instance* to a corresponding *target-FHIR-instance*.

The conversion of a *source-HHR-instance* starts applying the conversion rules represented by the tag <class> corresponding to the type of that instance.

In the next sections specify the syntax and semantics of the XML tags used by the HHR mapping language.

Tag <hhr-to-fhir>

The tag <*hhr-to-fhir*> is the root tag of any mapping document expressed with the HHR mapping language. It must contain one or more nested tags, chosen from <class>, <enum> or <role>.

Tag <class>

The tag <*class*> is a nested repeatable tag of the tag <hhr-to-fhir>. The tag <class> is used to represent a (source) HHR class and the corresponding (target) FHIR resource type or complex data type. The tag <hhr-to-fhir> contains an occurrence of this tag for each conceptual class of the HHR model.

It is possible to nest in the tag <class> zero or more tags <attribute>, <resource> or <categoryAttribute>. The mapping expressed by a tag <class> for an HHR class is also a default mapping for the attributes of all its HHR subclasses. Therefore, the mapping of an HHR class is expressed by the information contained in the tag <class> having the corresponding *hhrName* plus the mapping information expressed by the tag <class> or <role> of its superclass and recursively by its ancestor classes. The content of a tag <class> may override (i.e. substitute), other then extend, the mapping inherited from the superclass tag <class> or <role>.

The following tag-attributes are supported by the tag <class>:

- **hhrName**: the name of a HHR conceptual class that is the type of a *source-HHR-instance* that can be converted to a *target-FHIR-instance* using the mapping expressed by this tag. It's a syntax error if a mapping file contains two occurrences of tag <class> with a same value of the attribute *hhrName*. Note that the HHR model arranges classes into UML packages but it is forbidden to have two classes with the same name, also if they belong to different packages, therefore only simple HHR class names, without package names, are used as values of *hhrName*.
- [optional] **fhirName**: the name of the FHIR type (a resource type or a complex data type) of the corresponding *target-FHIR-instance*. By default, *fhirName* is equal to *hhrName*. If *fhirName* is set to the empty string, then the source HHR conceptual class have no corresponding FHIR type (but as the attributes of the source class are inherited by the subclasses they may be still mapped in the tags <class> of the subclasses).
- [optional] **hhrSupertype**: the name of the HHR conceptual class that is the superclass of the class referred by *hhrName*, if any (note that in the HHR model only single inheritance is allowed).
- [optional] **isAbstract**: ‘true’ if the HHR conceptual class named *hhrName* is abstract, ‘false’ otherwise.

Tag <role>

The tag <*role*> is a nested repeatable tag of the tag <hhr-to-fhir>. This tag <role> is very similar to the tag <class> and is used instead of the tag <class> when the class to map has the UML stereotype <<role>>. The tag <hhr-to-fhir> contains an occurrence of this tag for each conceptual class of the HHR model having stereotype <<role>>. The tag <role> may have the same nested tags and tag-attributes of the tag <class>, having the same semantics and constraints. Note that any HHR <<role>> class must always have an attribute named "player", that must be mapped using a nested tag <attribute>.

Tag <attribute>

The tag <*attribute*> is a nested repeatable tag of the tags <class>, <role>, <instance>. Each tag <attribute> specifies a rule to convert and copy the values of an (possibly nested) attribute (*source-attribute*) of the *source-HHR-instance* into the values of a corresponding (possibly nested) attribute (*target-attribute*) of the *target-FHIR-instance* or to set the *target-attribute* to a predefined value. Tags <attribute> can be nested within other tags <attribute> to represent the conversion rules of attributes that have complex values (i.e. not primitive). Nested tags are useful when the conversion of complex values depends from the containing attribute. If some type of complex value is mapped also with a tag <class> or <role>, the mapping expressed by the containing attribute overrides the one expressed by the tags <class> or <role>.

If the values of the *source-HHR-instance* are primitive, they are copied into the corresponding attribute of the *target-FHIR-instance* exactly as they are, without any conversion. If the values of the attribute of the source-*HHR-instance* are not primitive, they are firstly converted by means of the mapping rules corresponding to their type, expressed by the nested tags <attribute> (if any) or (if no nested tags <attribute> are specified) by the tags <class> corresponding to the HHR type of the value, and then copied to the attribute of the *target-FHIR-instance*. If no tag <class> for the type of the values is specified, then also complex values are copied without any conversion. Note that the conversion of the values is not done on the base of the type of the attribute but on the base of the type of the value itself.

The mapping expressed by a tag <attribute> may override the one expresses in the tag <class> of the HHR superclass and may be overridden by a corresponding tag <attribute> in the tag <class> of an HHR subclass.

The following tag-attributes are supported by the tag <attribute>:

- [optional] **hhrPath**: the path, relative to the *source-HHR-instance,* of the *source-attribute* to convert to a corresponding *target-attribute* of the *target-FHIR-instance*. The *hhrPath* is expressed as a *FHIRPath* expression. If two tags <attribute> within the same containing tag specifies a same *hhrPath* they must specify a different *hhrType* (see tag-attributes *hhrType* and *isMultipleValue*) or must specify a different *fhirPath*. Note that a *FHIRPath* expression may identify a set of values to convert that belong to different objects, e.g. if *hhrPath*="x.y" then the conversion and copy is applied to any value of the attribute y of any value of the attribute x of the *source-HHR-instance*. The *hhrPath* may also be set to an empty string. In this case the tag <attribute> is used to set the value of a FHIR *target-attribute* with a predefined value (see tag-attribute *fhirValue*) or with a complex FHIR value defined by nested tags <attribute>. By default, *hhrPath* is equal to an empty string.
- [optional] **fhirPath**: the path, relative to a *target-FHIR-instance,* of the *target-attribute*, expressed using the *FHIRPath* syntax. This path normally identifies a single node (i.e.

attribute) within the tree structure of the *target-FHIR-instance*. If the *fhirPath* identifies more than one node, then the converted values (i.e. the results of the conversion of the values of the attribute identified by *hhrPath*) are set as values of each identified FHIR node. Moreover, if a path $x_1.x_2. …x_n$ is specified and for some integer i<n the path $x_1.x_2. …x_i$ refers to a not valorised FHIR attribute x_i, then a new (empty) FHIR entity V_i is set as value of that x_i, in order to assure the full path is actually traversable. The type T_i of V_i is explicitly specified by the *fhirPath* using the *FHIRPath* syntax for polymorphic items, i.e. $x_1.x_2. …x_i(T_i). …x_n$, or is otherwise assumed to be the same type of the attribute x_i. If the type T_i determined with these rules is not instantiable, then a conversion exception is raised. Several tags <attribute> having the same *fhirPath* may be specified within the same tag <class> or <role> to assign more than one value to a same *target-attribute*. If no *fhirPath* is explicitly specified, then it is assumed to be equal to the hhrPath. If the *fhirPath* is explicitly set to an empty string (i.e. *fhirPath*="") then the *source-attribute* have no corresponding FHIR *target-attribute* (but it is possible, nesting other tags <attribute>, to map the attributes of the values of the *source-attribute* to corresponding FHIR *target-attribute*). An exception is raised if both *hhrPath* and *fhirPath* are set to an empty string.

- **hhrType**: the name of the type of the *source-attribute*. If two tags <attribute> within the same containing tag specify a same *hhrPath* they must specify a different *hhrType*. An HHR attribute is allowed to have only values of a type specified by one of the tags <attribute> having the *hhrPath* of that attribute. Moreover, more than one type may be specified as value of *hhrType* using the pipe (|) to separate the different types.

- [optional] **fhirExtension**: the *name* of a FHIR extension used as *target-attribute*. By convention, the fully qualified name of the FHIR StructureDefinition defining the specified extension is http://www.crowdhealth.eu/fhir/StructureDefinition/*name*, where the *name* is the one specified by *fhirExtension*.

- [optional] **fhirValue**: used to set the value of the *target-attribute,* when it does not depend on a *source-attribute* (i.e. when *hhrPath=""*). Using the tag-attribute *fhirValue* the *target-attribute* may be set to a fixed data value or to the *target-FHIR-instance* itself, represented by the special expression $this. The tag-attribute *fhirValue* cannot be used if *hhrPath* is not equal to "".

- [optional] **isMultipleValue**: 'true' if the *source-attribute* indicated by the *hhrPath* accepts more than one value of the specified *hhrType*. If no *hhrType* is specified, the constraint applies to values of any type. The default value of *isMultipleValue* is 'false'. This tag-attribute cannot be used if the *hhrPath* is empty (*hhrPath=""*). Note that if two tags <attribute> with the same *hhrPath* have *isMultipleValue*="false" the *target-attribute* will be allowed to have two distinct values (one for each different *hhrType*).

Tag <categoryAttribute>

The tag <*categoryAttribute*> is a repeatable nested tag of the tags <class>, <role>, <instance>. The tag <categoryAttribute> is used when the values of the (FHIR) *target-attribute* (e.g. the FHIR attribute named "category") depend on the type of the values of the (HHR) *source-attribute*. Note that this is different from the tag <attribute> where the values of the *target-attribute* depend from the values of the *source-attribute* instead than the type of the values. The tag <categoryAttribute> can be used also if the *target-attribute* has a fixed code as value. The usage of <categoryAttribute> is allowed only if the type of the *target-attribute* is *CodeableConcept* or *code*. The *target-attribute* is set to a fixed value using the tag-attribute *fhirCode*, or to values that depend on the type of the *source-attribute* (e.g. the attribute *type* of the HHR class *Procedure*) using the tag-attribute *hhrPath*. The value of the tag-attribute *fhirCode* cannot be set if the value of the tag-attribute *hhrPath* is set. The values of the attribute specified by the *hhrPath* must be instances of an enumeration and are converted to corresponding FHIR code values using the mapping specified by the corresponding tag <enum> (see also tag-attributes fhirCategoryType, fhirCategoryCode, fhirCategorySystem and fhirCategoryDisplay in tag <enum>).

The following tag-attributes are supported by the tag <categoryAttribute>:

- [optional] **hhrPath**: the path, relative to the *source-HHR-instance*, of the *source-attribute*. The default value of *hhrPath* is the empty string. If a *hhrPath* is set to the empty string, there is no *source-attribute* and the *target-attribute* is set to the value specified by tag-attribute *fhirCode* (together with tag-attributes *fhirSystem* and *fhirDisplay*).
- [optional] **fhirPath**: the path, relative to the *target-FHIR-instance*, of the *target-attribute*. The value of this tag-attribute is interpreted in the same way of the homonymous tag-attribute of tag <attribute>. When no explicit *fhirPath* is explicitly specified the default value "category" is assumed (note that this is different from the default of the same tag-attribute of tag <attribute>). Several tags <categoryAttribute> with the same *fhirPath* may be specified within the same tag <class> or <role> to assign more than one value to the same *target-attribute*.
- [optional] **fhirCode**: the value of the 'code' field of the value of the *target-attribute*. If the *hhrPath* is set to an empty string, then it is mandatory to set the *fhirCode* value with a not empty string.
- [optional] **fhirDisplay**: the value of the 'display' field of the value of the *target-attribute*.
- [optional] **fhirSystem**: the value of the 'system' field of the value of the *target-attribute*.
- [optional] **hhrType**: the type name of the value of the *source-attribute*. It must be the name of an enumeration defined in the HHR model (see tag <enum>).

Tag <resource>

The tag <*resource*> is a repeatable nested tag of the tags <class>, <role>, <attribute>, <instance>. The tag <resource> is used when the conversion of a *source-HHR-instance* implies the creation of some other related new FHIR resource other than the *target-FHIR-instance*. Such a new resource is specified using the <resource> tag. It is similar to the tag <class> because it implies the creation of a FHIR resource, and as such it can nest one or more <attribute> and <categoryAttribute>. The tag <resource> requires to specify an ID, that must unique in the scope of the containing tag, to be used as a reference to the created resource in the containing tag. By convention, a reference to a <resource> is expressed in the form $ID, where ID is the value of the tag-attribute *id* of the tag <resource>.

The following tag-attributes are supported by the tag <resource>:

- **id**: the local unique ID used to refer to the new FHIR resource in the scope of its containing tag.
- **fhirType**: the name of the type of the new FHIR resource to instantiate.

Tag <enum>

The tag <enum> is a nested repeatable tag of the tag <hhr-to-fhir>. It is used to map each value (*source-HHR-instance*) of an HHR enumeration *(source-enumeration*) to a corresponding value (*target-FHIR-instance*) of a FHIR datatype (*target-datatype*). The *target-FHIR-instance* may also belong to a specific FHIR ValueSet. The values of the *source-enumeration* are specified by means of the tag <instance> nested in the tag <enum>. Moreover, the *source-enumeration* itself may be mapped to a specific code or category value, using the tag-attributes fhirCategoryCode, fhirCategorySystem and fhirCategoryDisplay.

The following tag-attributes are supported by the tag <enum>:

- **hhrName**: the name of the *source-enumeration*.
- **fhirValueSet**: the FHIR ValueSet that *the target-FHIR-instance* belongs to.
- **fhirName**: the FHIR datatype (e.g. code or CodeableConcept) of the *target-FHIR-instance*.
- [optional] **fhirCodingSystem**: the name of the *coding system* to be assigned to the “system” field of the *target-FHIR-instance*, when it is a FHIR CodeableConcept.
- [optional] **fhirCategoryCode**: used to specify a FHIR code associated to the *source-enumeration* itself.
- [optional] **fhirCategoryType**: used to specify the type of FHIR code (a ValueSet or a CodeableConcept) associated to the enum itself.

- [optional] **fhirCategorySystem**: used to specify the system of the code associated to the *source-enumeration* itself.
- [optional] **fhirCategoryDisplay**: used to specify the display name of the code associated to the *source-enumeration* itself.
- [optional] **isAbstract**: used to specify if the *source-enumeration* is abstract. The default value is false.

Tag <instance>

The tag <*instance*> is used within a tag <enum> to map a *source-HHR-instance* that is a value of an HHR enumeration to a corresponding *target-FHIR-instance*. Tags <attribute> may be nested within the <instance> tag to set the attributes of complex FHIR values.

The following tag-attributes are supported by the tag <instance>:

- **hhrName**: the name of the *source-HHR-instance*.
- **fhirCode**: the *target-FHIR-instance* (when it is a FHIR "code"), or the value of the "code" field of the *target-FHIR-instance* (when it is a FHIR CodeableConcept).
- **fhirCodeDisplay**: the value of the "display" field of the *target-FHIR-instance*, when it is a FHIR CodeableConcept.
- [optional] **fhirCodingSystem**: the name of the coding system to be assigned to the "system" field of the *target-FHIR-instance*, when it is a FHIR CodeableConcept. When present this tag-attribute overrides (i.e. substitute) the value specified by the tag-attribute *fhirCodingSystem* of the containing tag <enum>.
- [optional] **fhirType**: the resource type or the datatype of the *target-FHIR-instance*. When this tag-attribute is present, it overrides (i.e. substitute) the value of the tag-attribute *fhirName* of the containing tag <enum>.

2.3. HHR to FHIR mapping example

This section reports an example of usage of the HHR mapping language. In particular, it shows how to map the class *Condition* of the HHR model. It maps the HHR class *Condition* to the homonymous FHIR resource type *Condition.* An instance of HHR *Condition* is converted to an instance of FHIR *Condition* plus a related FHIR instance of type *Provenance* (described by the tag <resource>), which attribute *target* is not mapped to any HHR attribute and is set to a reference to the FHIR *Condition* (expressed by setting the tag-attribute fhirV*alue="$this"*).

Note that, when a class (like *Condition*, in this example) inherits from a supertype (*Event*, in this example), the mapping of all the inherited attributes may be specified by the mapping of the supertype. If the supertype inherits from another supertype the mapping is also specified by the mapping of its supertype (if any) and so on. The specification of the mapping of an

HHR class ends where there is a type without any supertype. In the case of the class *Condition* of the HHR model, the mapping to FHIR ends in the class *IdentifiedEntity* which hasn't any supertype (the inheritance chain is *IdentifiedEntity*, *Event*, *Condition*).

The attribute *identifier* of the HHR class *Condition* is mapped to the attribute *identifier* of the FHIR type *Condition*. The mapping of this attribute is contained in the tag <class> of the HHR conceptual class *IdentifiedEntity*. Note that this HHR conceptual class have no correspondent FHIR type (indeed the tag-attribute *fhirName* is empty). More in details the value of the *value* attribute of *Identifier* is set to the *value* attribute of FHIR identifier while the attribute *system* of the HHR class *Identifier* is set with the value of the attribute *system* of the FHIR type *Identifier*. Note that *isMultipleValue*="true" so there can be more than one value for the attribute *identifier*.

The attribute *recorder* of type Agent is mapped to the attribute *agent.who* of the FHIR type *Provenance* defined in the tag <resource> having *ID* 'prov' (i.e. *fhirPath*="$prov.agent.who"), while the HHR attribute *recorderWhen* is mapped to the attribute *recorder* of the same FHIR instance of type *Provenance* (fhirPath="$prov.recorded").

The attribute *isAutomatic* of the HHR class *Condition* is mapped to an extension of the FHIR type *Condition*, which *StructureDefinition* has URI http://www.crowdhealth.eu/fhir/StructureDefinition/is-automatic.

The attribute *subject* of the HHR type *Condition* is mapped to the homonymous attribute of FHIR type *Condition*.

The attribute *asserter* of HHR *Condition* is also mapped to the homonymous attribute of FHIR Condition. According to the HHR model reported in D3.1, the *asserter* can be a Patient or a Practitioner (Patient and Practitioner are both subclasses of HealthCarePerson).

The HHR attribute *performedWhen* is mapped to an extension (of type *Period*) of the FHIR type *Condition*, which StructureDefinition is defined at http://www.crowdhealth.eu/fhir/StructureDefinition/performed-when.

The attribute HHR *assertedWhen* is mapped to the FHIR attribute *assertedDate* (of type Period).

The HHR attribute *conditionClinicalStatus* is mapped to the FHIR attribute *clinicalStatus* and it can be set with one of the values listed in the mapping of the *ClinicalStatus* enum.

The HHR attribute *subjectAge* is mapped to the FHIRattribute *onset*. It can be a Range or a Period depending on the kind of value set in *subjectAge*.

If the value of *conditionType* is an instance of the HHR enumeration *ClinicalFinding, then* the attribute *category* of the FHIR Condition is set to a *CodeableConcept* which field *system* is equal to 'http://www.crowdhealth.eu', which field *code* is equal to 'clinical-finding' and which field *display* is equal to ‘Clinical finding’. If the value of *conditionType* is an instance of the HHR enumeration *Diagnosis*, the field *system* is equal to ‘http://www.crowdhealth.eu’, the filed *code* is equal to ‘diagnosis’ and the field *display* is equal to ‘Diagnosis’.

The HHR attribute *type* is mapped to the FHIR attribute *code* of the target FHIR Condition. The type is *CodeableConcept* and the values of the fields *system*, *code*, and *display* depend on the type of the value (in this case the enum *Diagnosis* or *ClinicalFinding*).

Finally, the HHR attribute *performer* is mapped to an extension of the FHIR type *Condition*, which *StructureDefinition* has URI [http://www.crowdhealth.eu/fhir/StructureDefinition/performer.](http://www.crowdhealth.eu/fhir/StructureDefinition/performer) The value is a reference to an instance of FHIR *Patient* or FHIR *Practitioner*. Note that isMultipleValue=”true” so there can be more than one instance of the attribute *performer*.

```
<hhr-to-fhir>

    <class hhrName="IdentifiedEntity" fhirName="" isAbstract="true" >
        <attribute hhrPath="identifier" hhrType="Identifier" isMultipleValue="true" />
    </class>

    <class hhrName="Identifier">
        <attribute hhrPath="value" hhrType="string" />
        <attribute hhrPath="system" hhrType="string" />
    </class>

    <class hhrName="Event" fhirName="" hhrSupertype="IdentifiedEntity" isAbstract="true">
        <resource id="prov" fhirType="Provenance">
            <attribute fhirPath="target" fhirValue="$this" />
        </resource>
        <attribute hhrPath="recorder" fhirPath="$prov.agent.who" hhrType="Agent" />
        <attribute hhrPath="recordedWhen" fhirPath="$prov.recorded" hhrType="dateTime" />
        <attribute hhrPath="isAutomatic" fhirExtension="isAutomatic" hhrType="boolean"/>
    </class>

    <class hhrName="Condition" hhrSupertype="Event" isAbstract="true" >
        <attribute hhrPath="subject" hhrType="Patient" />
        <attribute hhrPath="performer" fhirExtension="performer" hhrType="HealthCarePerson" isMultipleValue="true" />
        <attribute hhrPath="asserter" fhirPath="asserter" hhrType="HealthCarePerson" />
        <attribute hhrPath="performedWhen" fhirExtension="performedWhen" hhrType="Period" />
        <attribute hhrPath="assertedWhen" fhirPath="assertedDate" hhrType="dateTime" />
        <attribute hhrPath="conditionClinicalStatus" fhirPath="clinicalStatus" hhrType="ClinicalStatus" />
        <attribute hhrPath="subjectAge" fhirPath="onset" hhrType="Range|Duration" />
        <categoryAttribute hhrPath="conditionType" hhrType="ConditionType" />
        <attribute hhrPath="type" fhirPath="code" hhrType="ConditionType" />
    </class>

    <enum hhrName="ConditionType" isAbstract="true"/>

    <enum hhrName="ClinicalStatus" fhirName="code" fhirValueSet="condition-clinical">
        <instance hhrName="ACTIVE" fhirCode="active" />
        <instance hhrName="ACTIVE_RECURRENCE" fhirCode="recurrence" />
        <instance hhrName="INACTIVE" fhirCode="inactive" />
        <instance hhrName="INACTIVE_REMISSION" fhirCode="remission" />
        <instance hhrName="INACTIVE_RESOLVED" fhirCode="resolved" />
    </enum>
```

<enum hhrName="Diagnosis" fhirName="CodeableConcept" fhirCodingSystem="http://www.crowdhealth.eu/hhr-t" hhrSupertype="ConditionType" fhirCategoryType="CodeableConcept" fhirCategoryCode="diagnosis" fhirCategoryDisplay="Diagnosis" fhirCategorySystem="http://www.crowdhealth.eu/hhr-t">

<instance hhrName="INTRADUCTAL_CARCINOMA" fhirCode="intraductal-carcinoma" fhirCodeDisplay="Intraductal carcinoma" />
<instance hhrName="ESTROGEN_RECEPTOR_POSITIVE_TUMOR" fhirCode="estrogen-receptor-positive-tumor" fhirCodeDisplay="Estrogen receptor positive tumor" />
<instance hhrName="ESTROGEN_RECEPTOR_NEGATIVE_NEOPLASM" fhirCode="estrogen-receptor-negative-neoplasm" fhirCodeDisplay="Estrogen receptor negative neoplasm" />
<instance hhrName="PROGESTERONE_RECEPTOR_POSITIVE_TUMOR" fhirCode="progesterone-receptor-positive-tumor" fhirCodeDisplay="Progesterone receptor positive tumor" />
<instance hhrName="PROGESTERONE_RECEPTOR_NEGATIVE_NEOPLASM" fhirCode="progesterone-receptor-negativeneoplasm" fhirCodeDisplay="Progesterone receptor negative neoplasm" />
<instance hhrName="POSITIVE_CARCINOMA_OF_BREAST" fhirCode="her2-positive-carcinoma-of-breast" fhirCodeDisplay="HER2-positive carcinoma of breast" />
<instance hhrName="HUMAN_EPIDERMAL_GROWTH_FACTOR_2_NEGATIVE_CARCINOMA_OF_BREAST" fhirCode="" fhirCodeDisplay="Human epidermal growth factor 2 negative carcinoma of breast" />
<instance hhrName="MALIGNANT_TUMOR_OF_BREAST" fhirCode="malignant-tumor-of-breast" fhirCodeDisplay="Malignant tumor of breast" />
<instance hhrName="SECONDARY_MALIGNANT_NEOPLASM_OF_LIVER" fhirCode="secondary-malignant-neoplasm-of-liver" fhirCodeDisplay="Secondary malignant neoplasm of liver" />
<instance hhrName="SECONDARY_MALIGNANT_NEOPLASM_OF_LUNG" fhirCode="secondary-malignant-neoplasm-of-lung" fhirCodeDisplay="Secondary malignant neoplasm of lung" />

</enum>

</hhr-to-fhir>

3. Source code

The current prototype of the HHR Manager is released on the *Artefacts* repository of the project as a jar file named “HHR Manager_v1.0.0”, while the machine-interpretable mapping is released as a separate XML file named “hhr_to_fhir_v1.0.0.xml”. The jar file contains the source code of the HHR manager written in Java 8. The mapping file is written in XML version 1.0.

3.1. Availability

Both the files can be downloaded from the project repository:
https://crowdhealthtasks.ds.unipi.gr/CrowdHEALTH/artefacts/tree/master/HHRManager.

3.2. Usage

The HHR Manager have no dependencies, apart the availability of a standard java virtual machine that support Java 8, and can be imported in any compatible project. Similarly, the mapping file may be read with any XML parser compatible with XML version 1.0.