HeD Editor – Developer's Guide

The source code of the HeD editor is available on github at https://github.com/sharpc2b

All the code is available under the Apache v2 License
(ASLv2 - http://www.apache.org/licenses/LICENSE-2.0.html)

Arizona State University and Intermountain Healthcare participated in the development of the first version of the editor. Users and developers and encouraged to use the editor and contribute back with feedback, issue reports, feature requests, improvements, bug fixes or any way that can help improve the quality of the product.

This guide is meant to provide an overview of the source code and its design principles.

The editor is written in Java, Javascript and Scala. The data is modelled using XML, JSON and OWL. The editor's code compilation requires Apache Maven. The runtime requires the "Play" Framework and a web application server such as Tomcat.
Developers are assumed to be familiar with these technologies. If not, the reader should consult the relative documentation, which is freely available on the web.

Any comment or request can be posted through GitHub, or sent directly to davide.sottara@asu.edu

The HeD editor allows to author clinical decision support (CDS) knowledge artifacts in a way compatible with the Health eDecisions (HeD) standard.

The editor has the following features:

- A Java/Scala back-end application, based on the Play Framework, compatible with web and cloud deployment
- A Javascript UI client
- A set of OWL ontologies defining the internal model
- Integration with HermiT and JBoss Drools for inferencing and validation
- A modular architecture to facilitate extensions and integration with other systems

Architectural overview

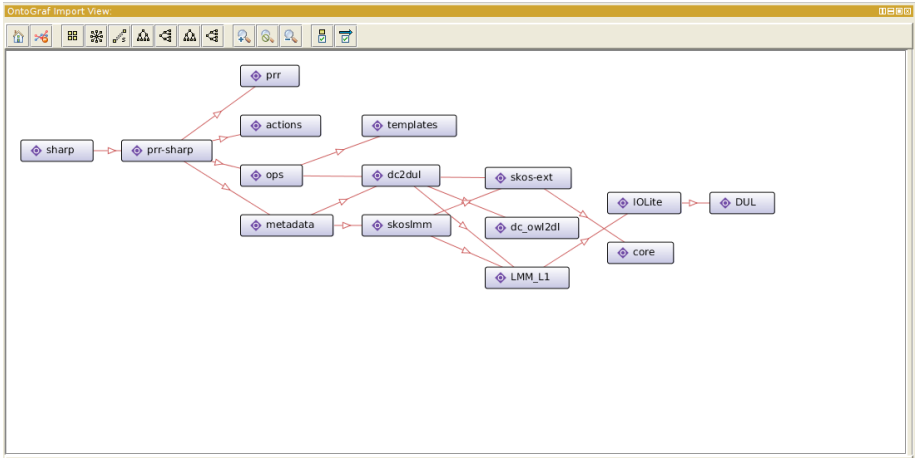
The editor's internal model is based on Description Logic, a widely adopted formalism with descriptive and inferential capabilities. We have chosen the Web Ontology Language v2 (OWL2-DL), a W3C standard designed for interoperability over the web. This choice facilitated the development of the models and the software, and its grounding in the context of existing "upper ontologies". The content of the HeD schema was inspired by some of the same models we have included or generalized in the first place. In particular, the foundations of our work are as follows. Credits are given to the authors of the original works:

- SKOS http://www.w3.org/TR/skos-primer, which was used to conceptualize clinical and medical terms and vocabularies.
- Dublin Core (DC) vocabulary http://dublincore.org, which was the basis for the HeD metadata.
- Production Rule Representation OMG standard (PRR) http://www.omg.org/spec/PRR/1.0, which provided the general structure of a Knowledge Artifact.
- LMM http://ontologydesignpatterns.org/wiki/Ontology:LMM, to capture concepts and the ability to reference and mention them, as well as the DULCE/IO-Lite ontologies, which allowed to contextualize our required concepts and provide support for a future integration of SSFs.

We have extended and harmonized these ontologies to include the specific concepts needed to model HeD artifacts and their content. In particular, we have added the following ontologies:

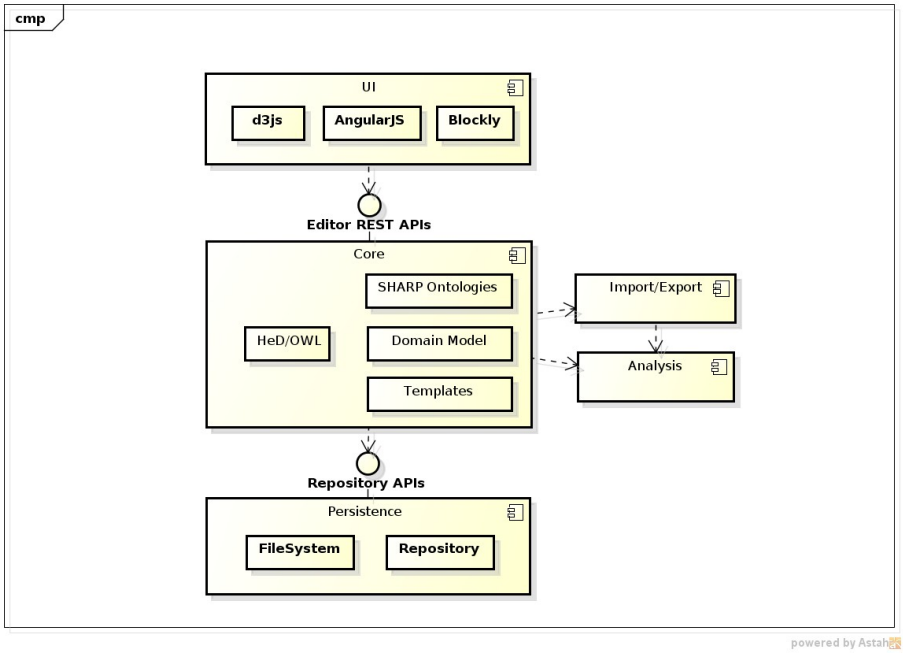
- An HeD "operation" ontology to model the expression language. The ontology can be extended to accommodate operations typical of an Object Constraint Language (OCL) which are not explicitly listed in HeD
- An extended metadata ontology, to capture clinical metadata which are not included in DC
- An "action" ontology to model the domain-specific recommendations and behaviors typical of a CDS artifact.
- A "template" ontology to describe common patterns in a CDS triggers, conditions and/or recommendations
- Some bridge ontologies to align and integrate the various modules.
- One or more "domain" ontologies based on the data model/vocabularies used in the artifacts. Currently the editor supports HL7 vMR, with plans to include HL7 FHIR in the near future.

The ontologies can be edited using Protege 4.x. Figure 1 shows the dependencies between the ontologies.



Editor internals

The ontologies are the models driving the editor, which in turn is based on a simple 3-tier architecture, shown in Figure 2. The persistence layer allows to store and retrieve KA from a repository (currently a simple repository based on a file-system implementation is provided, but APIs will allow to replace it with a more robust implementation). The KAs are stored in RDF format rather than HeD/XML, to preserve the additional information in the semantic description. The editor core is responsible for loading the artifact being authored and the ontologies required to model it. The core will also analyze the artifact, generate the internal data structures required during the authoring process and apply the additions and transformations requested by the user through the user interface. The presentation layer is a pure web-based application, written in Javascript, which interacts with the core through a set of REST APIs. The core is packaged as a Play™ application, which allows it to be deployed on the cloud, as well as a web application container such as Tomcat



Editor Development and Installation Guide
(It is assumed that the reader is familiar at least with Apache Maven, Git, the Play framework

Dependencies:

The entire project relies on Apache Maven 3.x to automate the build process. Maven is used to manage the various 3rd party library dependencies.
The major dependencies are as follows:

- Maven 3.x http://maven.apache.org/
- OWL-API 3.4.8 http://owlapi.sourceforge.net/
- Empire 0.7.3 https://github.com/mhgrove/Empire
- JaxB 2.2.5 https://jaxb.java.net/2.2.5/
- Sesame 2.7.0 http://www.openrdf.org/
- JBoss Drools 5.6.Final http://drools.jboss.org/
- Drools-Shapes 0.5.6.Final https://github.com/droolsjbpm/drools-chance
- HermiT 1.8.3 http://hermit-reasoner.com/
- Play Framework 2.1.0 http://www.playframework.com/

Runtime Dependencies, not required to build the editor's source code, are as follows:

- A running CTS II server
- CTS2 framework 0.8.4 http://informatics.mayo.edu/cts2/framework/

Optional Dependencies, not strictly necessary to run the editor, are as follows:

- Apache Stanbol https://stanbol.apache.org/
- Stanbol Client v 0.20 https://github.com/zaizi/apache-stanbol-client

Local Dependencies: a few libraries are not currently available on public Maven Servers, either because they are not Maven-enabled or because we have made some customizations which have not been included in an official release. For the time being, the libraries are uploaded in the */lib* folder. As the libraries become publicly available, we will update the POM files and the lib folder accordingly.

Editor Modules
The editor is composed of different modules, to help separate the various architectural elements. Maven manages the dependencies between the modules automatically.

- **Design Docs**

A growing collection of specifications, documentation and related resources. It is also the module where this document is hosted.

- **Editor Core**

The core is the main controller component, which relies on the editor models and orchestrates the ancillary components such as the persistence and import/export capabilities.

- **Editor Models**
  - *HeD Ontologies* – This module contains the base HeD ontologies, as described previously in this guide. The ontologies are serialized in one of the standard OWL formats and can be processed with the OWL APIs. The purpose of this module is to make the ontologies shareable and redistributable
  - *Generated Models* – This module contains the additional resources needed to build the editor's internal model. The resources are typically processed using the editor plugins (see the dedicated module's description) to generate the derived artifacts. The artifacts are generated in the target/generated-sources folder and the installed in the modules which require them at compile and run time. In particular, the "source" resources include:
    - The HeD XSD schema
    - The XSD schema of any domain model supported by the editor (currently: vMR)
    - XSD schemas describing target languages (e.g. Arden Syntax)
    - The Spreadsheet where the supported expressions and their admissible input and output types are defined. The XLS can be generated automatically from the specific source file (*expression.xsd*). However, it requires manual editing since the XSD does not include all the necessary information. The XLS format facilitates this editing process. The XLS is then processed by another plugin to complete the HeD expression OWL ontology.
    - The Spreadsheet where the editor templates are defined. The spreadsheet has been derived and is compatible with the specification of the HeD UC2 templates.

The information contained in the "source" resources is used to automatically generate the "target" resources, as follows:

- A JaxB – derived Java implementation of the classes in the HeD schema
- The OWL representation of the HeD expressions supported by the editor, as well as the logic to translate the expression from XML to OWL.
- The OWL/SKOS representation of the domain information model
- The Blockly descriptors of the HeD expressions and domain model classes and properties

  - *HeD Model* – This module is solely responsible for the generation of the Java classes and interfaces derived from the core HeD ontologies. The classes allow for an object-oriented manipulation of the concepts and their instances. The generated classes support both serialization and de-serialization to and from XML, RDF and XML.

- *HeD Model Expressions* – This module is solely responsible for the generation of the Java classes and interfaces corresponding to the HeD expression ontology. Given the large number of classes, the generation process has been split in two parts. This module depends on (and the classes extend) the code generated in the HeD Model module.

- *HeD Model Templates* – This module generates the classes required by the HeD templates. The logic also implements the process to load and integrate the templates into the editor core.

- *Model Analysis* – This module, currently being developed, contains the validation, correction and completion logic used by the editor to verify the syntactic and semantic correctness of the artifacts.

- *Arden Model* – This module generates the Java classes and interfaces to manipulate Arden artifacts according to the version 2.8 of that standard. The Arden export functionality has not been yet implemented to this date (May 2014)

- **Editor UI**

The editor UI is a web application, based on Javascript/Json. It is a combination of dynamic HTML pages and dynamic controllers. The UI uses AngularJS for the controllers, and D3JS for charting and advanced visualizations.

- **HeD Services**

This module contains the Play framework wrapper for the HeD editor. It defines the public APIs which allow the UI (and potentially other external clients) to interact with the editor core and the services it provides. In detail, the *routes* configuration file defines the REST URLs where the services are exposed. The routes are mapped to the "Controller" classes, which serialize the input and output parameters to and from JSON. The actual business logic is delegated to the *ModelHome* controller, which interacts with the Editor Core.

- **Import / Export**

This module contains the logic to convert an HeD XML artifact into its equivalent OWL representation. It also contains the logic to export back artifacts to HeD/XML, HTML, RDF/XML. As additional formats become supported, they will/should be added here.

- **Editor Persistence**
  - *Persistence API* – This modules defines the API required by any persistence layer implementation
  - *Persistence Factory* – This module is responsible for the discovery, selection and instantiation of the required persistence layer implementation.
  - *Filesystem Persistence* – This module implements a simple, file-system based persistence layer. The artifacts will be saved in OWL/RDF in a folder on the file system.
  - *Stanbol Persistence* – This module (experimental and under development) integrates the semantic content management system Stanbol, using it as an artifact repository.

- **Editor Plugins**

The plugins are used during the build process. While independent and reusable, their main role is to

help generate the derived artifacts:

- *Model Transform* – This module contains the logic to implement the transformation facilities provided through the plugins:
  - The generation of the HeD expression OWL ontology from the XLS spreadsheet. The logic also enables the generation of the Blockly descriptors and the XML/OWL import/export. Since the expressions are potentially not known in advance, or it may be desirable to extend the set of supported expressions in the future, the logic in this module tries to generate the expression model automatically, minimizing the amount of manual interventions.
  - The conversion of an OWL T-Box into a SKOS/OWL A-box. The source models a taxonomy (graph) using classes, subclasses and properties. The target uses individuals, instance of the skos:Concept class, to model the content of the former, and relies on SKOS-defined relationships (e.g. skos:broader) to connect them. The inverse process is also supported, while not currently used for the editor's purposes.
  - The parsing of the HeD template spreadsheet, to generate the HeD template ontology and integrate it with the HeD core ontologies

- *Define Operators Plugin*
- *Skos to TBox Conversion Plugin*
- *TBox to ABox Conversion Plugin*
- *Template Generation Plugin*

- **Editor Utils**
  A small collection of general-purpose helpers, shared by different modules and factored out for convenience.

Installation and Execution

1) Run maven to build the project:
mvn clean install [-DskipTests=true]

2a) Run the Play Framework from the **hed-services** module folder
play
start

OR

2b) create a deployable Play WAR and install it in the application server
play war

3) deploy the **editor-ui** WAR file in the application server