Guidelines for Driving Application Functionality

Application-Driven Ontology Framework Overview

Many functionalities of the eagle-i SWEET curation tool are driven by directly consuming the eagle-i ontologies. For example, axioms in the ontology define the primary research resources for which the application collects data, control what fields of metadata are presented for a given resource type, define the order in which these fields are presented, and restrict what values are allowable. There are two primary types of axioms that are used to direct application behavior.

1. Logical domain and range restrictions on properties define which properties hang from which classes to drive metadata collection in the UI.
2. Application-specific annotation axioms support all other ontology-driven functionality. These custom annotation properties and instance values are used to define axioms according to defined patterns recognized by the application. All of these annotation axioms that govern application functionality live in an "application layer", i.e. owl files that are separate from the domain content of the ontology.

Below, we define the properties, and axiomitization patterns and prescribed conventions for their use to guide how the eagle-i application layer builds a user interface and captures data in the SWEET curation tool. But first we provide a glossary of concepts and terms that are essential for understanding the interplay between the eagle-I ontologies and applications.

Notation Conventions in this Document

- Property names are *italicized*.
- Class names are in 'single quotes'.
- Individual names are in `monospace` font.
- Important concepts and references to sections, tables, or figures are **bolded**.

Glossary of Concepts and Terms

Here we present concepts that are specific to our application, and terms used to refer to them. Concepts and terms established in semantic web / ontology communities and standards are taken to be known by users, and not defined here.

Concept 1: Domain vs Application Ontology Content

"Domain" content in the eagle-i ontologies includes all axioms representing general knowledge pertinent to the domain of research resources. This content is partitioned into a set of files referred to as the "domain layer" of the ontology, which comprise the publicly released ERO ontology module that is shared for community re-use.

"Application" content includes axioms meant to drive application functionality. This is not shareable domain knowledge, and not part of publicly released ERO ontologies. This content is partitioned into a set of files that is referred to as the "application layer" of the ontology.

See the Table of Annotation and Domain Layer Files for a list of IRIs, key features and content of files that comprise the domain and application layers.

Concept 2: Extended vs Core Domain Content (model / files / classes)

This is an artificial distinction made based on how MIREOTed content is stored and loaded into memory.

"Extended" content (files / taxonomies / terms) comes from five ontology imports (Uberon, GO, MP, Pato, DOID). Due to their size, these are stored in owl files separate from the ero.owl core file that holds all other domain content. Their content makes these files too large to be loaded into memory as the application runs, and are therefore indexed separately and dynamically called as needed. These files are also handled separately by build scripts and housed in separate files at the end of the build pipeline.

The descriptor "extended" may be used below to refer to files, taxonomies, or terms:

- "Extended files/modules" are these five import files and their associated app files
- "Extended taxonomies' are the class hierarchies these files contain
- "Extended terms" are classes/properties/individuals in these hierarchies

"Core" content includes all other domain content that lives in the ero.owl module. This includes terms defined natively in the ERO namespace (ERO_XXXXXXX), and content MIREOTed from other ontologies, such as OBI, SWO, SO, OCRE, etc.

The extended vs core distinction is important, as specific application annotations are needed to guide the software in finding axioms in the right files, as detailed in the **Table of Annotation Properties and Individuals** and in the **Task Workflow and Scenarios** below. The classification of each file in the eagle-i ontology as being extended vs core is indicated in the far right column of the **Table of Annotation and Domain Layer Files** .

Concept 3: Resource Classes (aka Instantiated Classes)

“Resource” classes represent entities considered to be research resources for which instances are created in the eagle-i data. Subtypes of this concept include "primary" resources that are cataloged and shared, "embedded" resources that are directly related (1:1) with a single primary resource instance, and "stubbed" resources that can be related to more than one primary or embedded resource instance (and also not cataloged or shared in eagle-i).

Class Type	Description	Examples
Primary Resource Classes	Represents the main resources types cataloged and shared as instances in the eagle-i data.	Examples include: 'antibody reagent', 'software', 'service offering', 'instrument', ‘cell line’, etc.
Embedded Resource Classes	Represents non-primary resources for which instances are created and that are linked 1:1 with a single resource instance. Embedded resources are not cataloged and shared by eagle-i.	Examples include: 'construct inserts' as embedded resource types for 'DNA constructs', 'phenotype annotations' as embedded resource types for 'organisms'.
Stubbed Resource Classes	Represents non-primary resources for which instances are created that can be linked to more than one resource instance. Stubbed resources are also not cataloged and shared by eagle-i.	Examples include: 'genetic alteration’ instances, which can be associated with both primary resources or other stubbed resources (such as a ‘cell line’ instance and a ‘human subject’), and 'human subject' instances, which can be linked to more than one ‘primary cell line' instance.

Concept 4: Referenced Taxonomy

Referenced Taxonomies are ontology hierarchies that hold non-resource classes used as values to describe resource attributes. In the data, the IRIs of referenced taxonomy classes are used directly as the objects of RDF triples about a resource instance. Instances of referenced taxonomy classes are not created in the eagle-i data.

A hierarchy becomes a referenced taxonomy when its root is annotated as belonging to the referenced taxonomy class group, and a referenced taxonomy IRI is present. The root of a referenced taxonomy hierarchy can be defined as the range of a property used to collect data about a resource type that is the domain of that property. This axiom in the ontology tells the application to present a field for the resource based on the property, and shows the reference taxonomy rooted at the class indicated as the range in a pick list for entering values of this property in the SWEET UI.

Referenced taxonomies can live in the ero.owl core, or in the extended ontologies. For example, the 'technique' referenced taxonomy lives in ero.owl and is used to record methods related to a given resource, while the 'material anatomical entity' referenced taxonomy that lives in the uberon.owl extended ontology file is used to record the origin of things like 'cell lines' or 'biological specimens'. "Core Reference Taxonomies" are those that live in the core ero.owl file (e.g. 'technique', 'organization', 'data format specification', 'measurement scale'). "Extended Reference Taxonomies" are those that live in an extended ontology file (Uberon, GO, MP, Pato, DOID), e.g. 'life cycle stage', 'material anatomical entity', 'disease'.

Ontology Structure

Finally, it is also important to have a basic understanding of the file import structure through with the complete eagle-i ontology is assembled in order to correctly place axioms driving application functionality. The **Table of Annotation and Domain Layer Files** describes key features and content of each file. Note that each owl file containing domain content is paired with an application file (-app.owl) that holds application axioms on domain classes/properties. See the **Ontology File Structure** page for more details and a diagram of the import chain.

Next: Annotation Properties and Individuals for driving the Application