System Overview

Introduction

eagle-i is a distributed platform for creating and sharing semantically rich data. It is built around semantic web technologies and follows linked open data pri nciples. In its current incarnation and operational deployment, eagle-i focuses on biomedical research resources. However, thanks to its ontology-centric architecture, the platform can be adapted to other domains.

The eagle-i platform comprises software components deployed at a participating institution (an _eagle-i node) _and a central search application. Figure 1 provides a high-level overview of the eagle-i software components of a node:

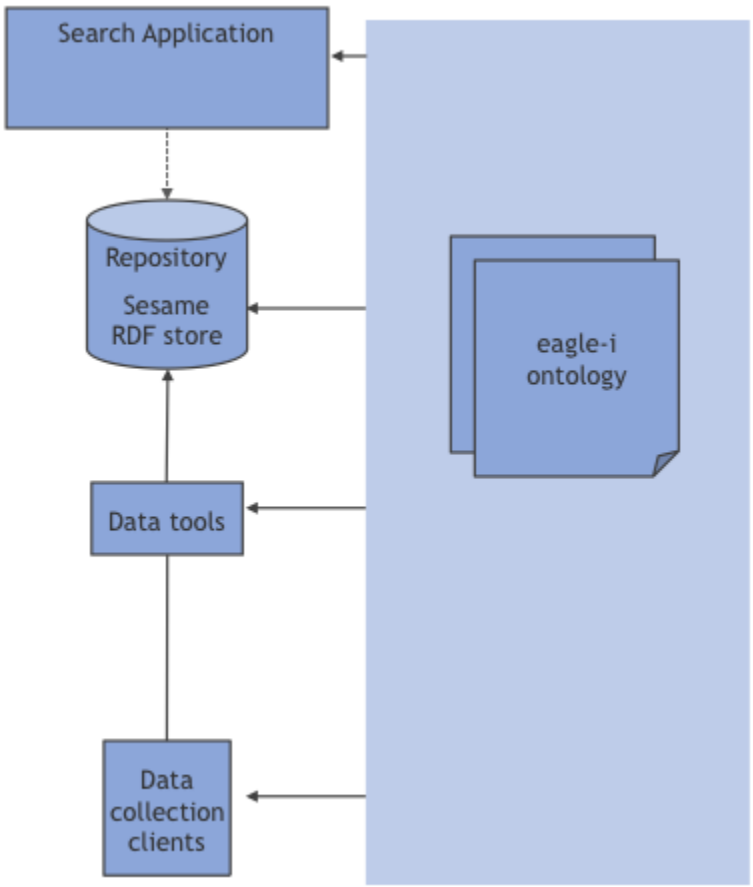

Figure 1. Overview of the eagle-i software stack

eagle-i resources are captured and stored as RDF data that conforms to a resource ontology. At the core of the eagle-i software stack is an RDF repository that provides services to institutional data collection applications, to an institutional search application and to the central search application. Data collection applications exist for both manual annotation and bulk data processing.

The eagle-i ontology drives the data collection and search user interfaces and internal mechanisms, and is used for structuring and validating data in the repository and for indexing resources. This design choice allows applications to seamlessly adapt to ontology evolution, and provides ontology developers with a mechanism to rapidly test and refine their models.

The eagle-i software stack

Figure 2 shows a more detailed view of the software stack components.

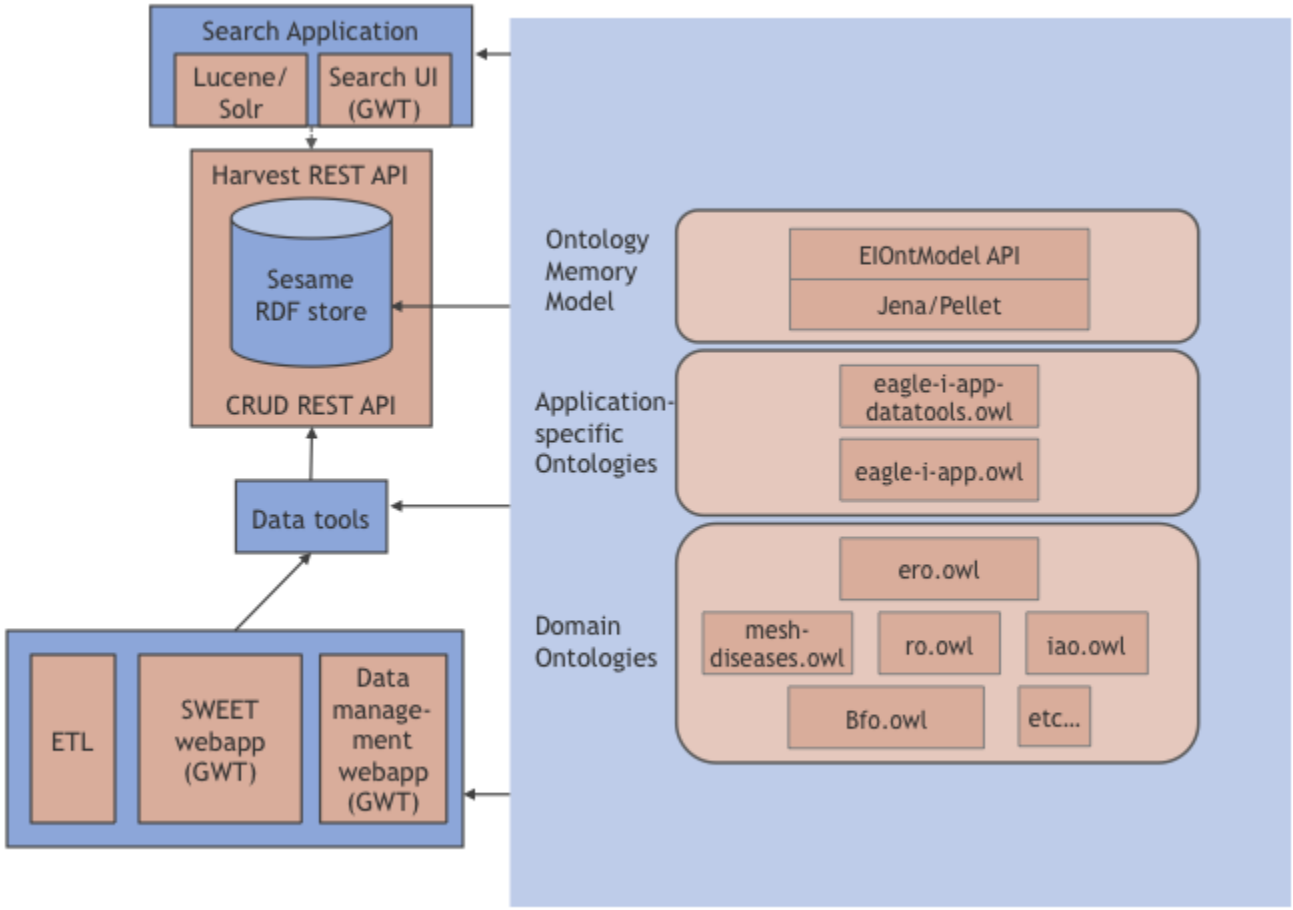

Figure 2. Details of the eagle-i stack

Repository

The eagle-i repository provides a REST API for storage and retrieval of eagle-i resource descriptions. It internally uses the Sesame RDF store. The repository functionality includes role-based access control, transactional CRUD operations on eagle-i instances [1], a data staging mechanism in the form of a curation workflow, a harvesting mechanism for incrementally communicating updates aimed at building search indices and a general purpose SPARQL query endpoint.

Data tools

The eagle-i platform utilizes a variety of tools for repository data ingest, broadly referred to as *data tools*. The different tools share a common layer for interacting with the eagle-i repository.

The **SWEET** (for *Semantic Web Entry and Editing Tool*) is a web application for manual data entry and curation developed using the GWT (Google Web Toolkit). Its core component is a dynamic forms generation module that translates ontology axioms into UI data entry widgets. The SWEET generates a form per ontology class and thusly allows users to create instances of an ontology class. Navigational elements of the SWEET include workflow controls and instance listing and filtering.

An **ETL (Extract Transform and Load) toolkit** provides command line tools for transforming Excel and XML data into eagle-i linked instances, and for loading them into an eagle-i repository. The data mappings necessary for performing the transformations are captured in an RDF map, in an approach heavily influenced by the RDF123 tool.

A **data management toolkit** provides command line tools for performing bulk modifications on instances in a repository, and in particular for migrating existing instance data to conform to a newly released ontology. A data management front-end to be used by data curators is under active development.

Search

The search application backend is a semantic search framework, where pluggable *search providers* perform searches against different types of indices and public services. A Solr search provider, the core backend component of the search application, builds a collection of Lucene/Solr indices that enable search and autocomplete functionality across the eagle-i dataset and ontology. The search backend also includes providers for a variety of external sources such as NCBI and NIF.

The search application front-end is a web application developed using the GWT (Google Web Toolkit). The search UI includes features such as faceted search, synonym expansion, autocomplete based on instance data and ontology terms.

Ontology/Data Model

The eagle-i ontology-centric architecture is based on an ontology layering approach, where *eagle-i application ontologies* annotate domain ontologies to implement constraints and convey information required by the eagle-i applications. This provides separation of concerns, such that domain ontologies can evolve without major impact on the applications.

The search application and the different data tools use a data model component, the EIOntModel, to obtain information about ontology classes and properties. The data model is optimized to be used by client-side applications.

In March 2012, the eagle-i ontology was officially released. For more in-depth information about it, see the google code project site.

The eagle-i network

In the eagle-i network, institutional servers are independent of each other. A central search application provides a unified view across the institutional servers.

We have experimented with a deployment where the central search application queries institutional search indices using SPIN, a federated query network that distributes queries and aggregates results. This provides maximal institutional independence, as institutions can join and pull out of the federation at any moment. However, with an heterogeneous data set such as eagle-i's, ranking search results is a non-trivial task.

We are currently using an alternate deployment whereby the search application builds central indices by harvesting data from the institutional repositories.

Figure 3 shows an eagle-i deployment using SPIN.

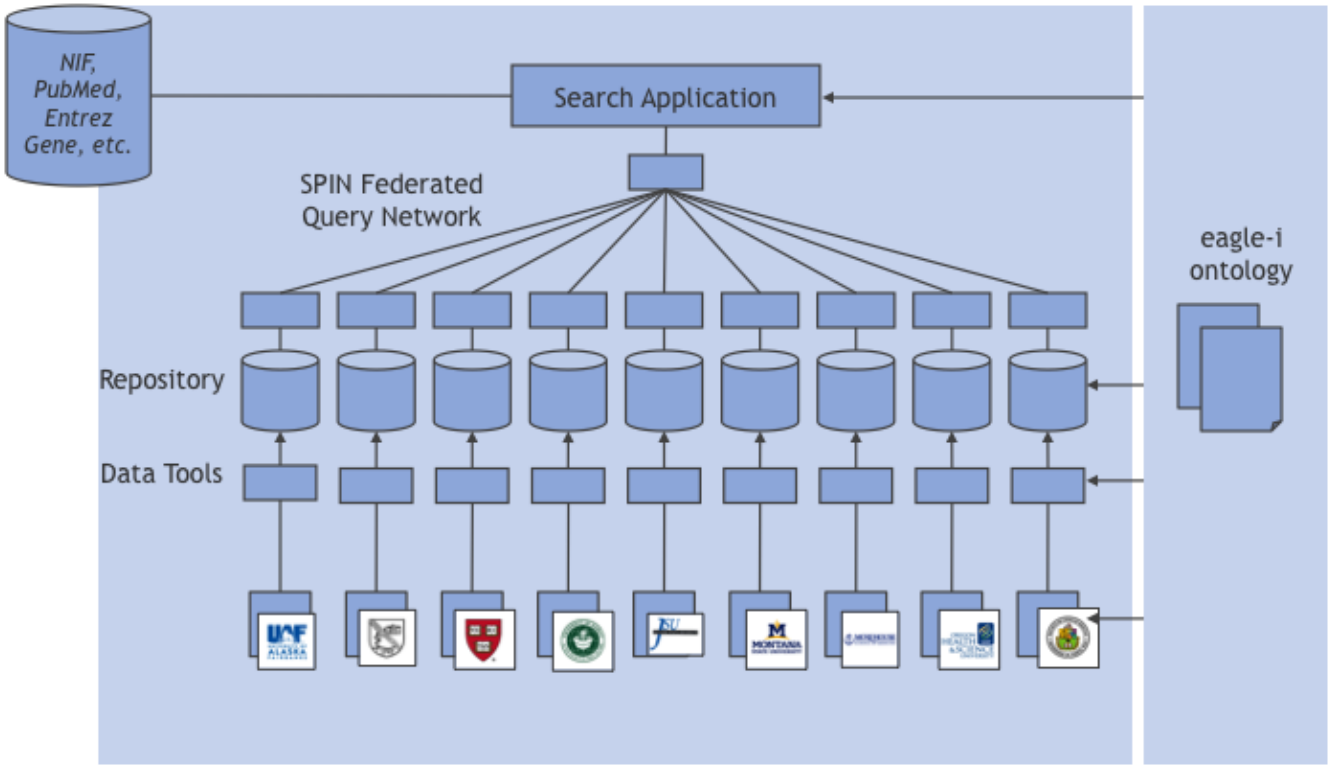

Figure 3. Sample eagle-i deployment

Terminology

[1] eagle-i resource instance - a collection of RDF statements about the same subject or about an *embedded instance* [2] subject, plus the display labels of all predicates and objects used in these statements.

[2] embedded instance - an instance that can only exist in the context of a parent instance and that cannot be linked to from other instances. Embedded instances do not have provenance metadata

How are we doing?

Is there anything that could be clearer in our documentation? We welcome your questions and feedback.