Product Search using Linked Data

John Walker[1] and Herman Elenbaas[2]

[1] Semaku, Torenallee 20, 5617 BC Eindhoven, Netherlands
john.walker@semaku.com
[2] Nexperia, Jonkerbosplein 52, 6534 AB Nijmegen, Netherlands
herman.elenbaas@nexperia.com

Abstract. Nexperia is a semiconductor manufacturer with a rich heritage from Philips and NXP. With such a long history, the company has built an impressive product portfolio of over ten thousand distinct product models. With this many products, finding the relevant data across the various systems can be an onerous task. To that end Nexperia have adopted and extended the award-winning Enterprise Data Hub [1] solution originally developed in cooperation with NXP and publish Linked Data on the Nexperia Data Portal [2]. One of the key new developments has been the Product Search application which enables users to quickly find answers to the most frequently asked questions from customers in a fast, responsive and intuitive interface. Typically, this might be a Nexperia sales representative talking to a customer who needs information about what products to purchase. Being able to provide fast and accurate product information can help Nexperia to win more sales. To ensure a fast search experience, we use an Elasticsearch cluster for the search index. In this paper we explain the Extract, Transform, Load (ETL) approach used to populate the search index and how Linked Data adds value in the Product Search application.

Keywords: RDF, SPARQL, JSON-LD, ETL, search.

1 ETL process

From previous work on the Product Data Hub, almost all the required data is available as RDF and can be queried via SPARQL. The approach taken in the Enterprise Data Hub is to maintain a clear relationship and provenance of data from the source systems across various domains and disciplines. To this end, there is a repository per source where the data is modeled in RDF maintaining a structure close to the representation in the source system. Where possible, common keys are already mapped to standard URI templates to already interlink data across sources, but this is not always possible. The use of RDF provides a normalized data layer where data can easily be blended and mashed up. A schema-on-read approach is applied, where SPARQL 1.1 Query and SPARQL 1.1 Federated Query are used to gather and integrate data from several repositories. The CONSTRUCT form of query is used to map onto the target model.

In earlier versions of the application we used monolithic SPARQL queries to extract a complete description of a product in a single request. As the complexity of the queries

increased, we saw a degradation in performance and maintainability of the queries. To remedy this, we decided to split the monolithic queries into several smaller CONSTRUCT queries where the results of the queries can be combined using an RDF merge operation to concatenate the graphs.

Not only is the performance of these smaller queries much more predictable, it also reduces the workload that the SPARQL endpoint must perform on a single request and therefore reduces the stress on the server. The queries are also significantly easier to understand for the developers and therefore easier to develop, debug and maintain. Additional benefits of doing this is it is now possible to test and measure performance of the individual queries to identify bottlenecks and take actions to improve performance. This also allowed us to change the ETL approach, which was previously an incremental approach product-by-product, to also enable a full reload of the search index in a performant manner.

The merge of the results from the queries is done in a Jena in-memory model. We then extract an unbounded sub-model from this per product resource. The sub-model is then framed using JSON-LD Framing to coerce the graph into a hierarchical idiomatic JSON structure and to alias URIs to developer-friendly 'local' names. These framed documents are then added to the Elasticsearch index. On a full reload of the index, we create a new index alongside the current index and swap these once the new index is populated. This gives zero downtime for users. The process for a full refresh takes under 5 minutes to complete.

2 Product Search application

The Product Search application was developed in close cooperation with business users. We followed a user-oriented approach with a design sprint to make sure we truly understand the user needs and requirements, resulting in a validated design. This was followed by several implementation sprints to bring the design to a working application.

The application consists of Java Spring Boot server-side and Vue.js client-side running in the web browser. The server mediates requests to the backend services by exposing an API consumed by the client. The backend services include calls to the Elasticsearch index, SPARQL queries and a third-party API for external stock and pricing data.

The application leverages the linked nature of the data by displaying links to other resources and by using the property URIs to lookup the labels and definitions of those properties for display in the application.

References

1. Enterprise Linked Data award 2015, https://2015.semantics.cc/eldc-awards-given, last accessed 2018/06/04
2. Nexperia Data Portal, http://www.data.nexperia.com/