Final Project Report

COORDINATED HOLISTIC ALIGNMENT OF MANUFACTURING PROCESSES (CHAMP)	
Principle Investigator / Email Address	Barry Smith – phismith@buffalo.edu
Project Team Lead	The Research Foundation for SUNY on behalf of the University at Buffalo
Project Designation	DMDII-15-11-03
UI LABS Contract Number	0220160023
Project Participants	CUBRC Inc
DMDII Funding Value	$626,223.00
Project Team Cost Share	$629,037.00
Award Date	December 30, 2016
Completion Date	June 30, 2018

SPONSORSHIP DISCLAIMER STATEMENT: This project was completed under the Cooperative Agreement W31P4Q-14-2-0001, between U.S. Army - Army Contracting Command - Redstone and UI LABS on behalf of the Digital Manufacturing and Design Innovation Institute. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Army.

DISTRIBUTION STATEMENT A. Approved for public release; distribution unlimited.

Contents

Acronyms and Abbreviations iv
1 Introduction 1
2 CHAMP Ontologies 3
2.1 Constraints 3
2.1.1 Web Ontology Language (OWL) 3
2.1.2 Basic Formal Ontology as Top-Level Ontology 3
2.1.3 Common Core Ontologies as Mid-Level Ontologies 4
2.1.4 Ontological Realism 6
2.2 Requirements 6
2.2.1 Consistency 7
2.2.2 Principle of Single Inheritance 7
2.2.3 Definitions 7
2.2.4 Preservation of Meaning of Higher-Level Ontology Terms 8
2.3 Accomplishments 9
2.3.1 The Industry Ontology Foundry 9
2.3.2 Education 9
2.3.3 The Product Life Cycle Ontologies 10
2.4 Lessons Learned 12
3 Working with Data: Alignment and Ingestion 13
3.1 OntoView 14
3.1.1 Overview 14
3.1.2 Requirements 17
3.1.3 Architecture 17
3.1.4 Accomplishments 18
3.1.5 Lessons Learned 18
3.2 Process Workflow 20
3.2.1 Overview 20
3.2.2 Requirements 22
3.2.3 Architecture 22
3.2.4 Accomplishments 23
3.2.5 Lessons Learned 23

3.3 Natural Language Processing 24
3.3.1 Overview 24
3.3.2 Lessons Learned 24
4 References 25

Table of Figures

Figure 1 The Common Core development methodology 5
Figure 2 The import structure among the seven ontologies of the product life cycle suite 11
Figure 3: OntoView landing page 14
Figure 4: OntoView taxonomy search 15
Figure 5: OntoView search result 15
Figure 6: OntoView data exploration breadcrumb 15
Figure 7: OntoView query builder 16
Figure 8: OntoView function editor 16
Figure 9: OntoView architecture 17
Figure 10: Process Workflow landing page 20
Figure 11: Process Workflow processes table 20
Figure 12: Process Workflow process editor 21
Figure 13: Process Workflow legacy data input landing page 21
Figure 14: Process Workflow legacy data input 22
Figure 15: Process Workflow architecture 22

Acronyms and Abbreviations

Acronyms	Terms
BFO	Basic Formal Ontology
CCO	Common Core Ontologies
CHAMP	Coordinated Holistic Alignment of Manufacturing Processes
CRUD	Create-Read-Update-Delete
GUI	Graphical User Interface
IOF	Industry Ontology Foundry
IRI	International Resource Identifier
JSON	JavaScript Object Notation
JSON-API	JSON Abstract Public Interface
MVC	Model-View-Controller Framework
NLP	Natural Language Processing
OWL	Web Ontology Language
RDF	Resource Description Framework
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language
URL	Uniform Resource Locator
URN	Uniform Resource Name
YAML	YAML Ain't Markup Language

1 Introduction

The University at Buffalo, CUBRC, and Cobham Industries, performed the Coordinated Holistic Alignment of Manufacturing Processes (CHAMP) project with the objective of enabling manufacturing organizations to overcome some of the issues caused by data heterogeneity. In particular, the CHAMP project sought to provide remedies to situations in which disparate data sources cannot be used in combination because of differences in their underlying conceptual schemas. This semantic heterogeneity is often cited in research in the model based development community as having deleterious effects on the fruition of the objectives of model based engineering.

At the organizational level, the introduction of semantic heterogeneity is due, as described in one of the Gartner research articles on the "Postmodern ERP" [4], by the shift away from single vendor ERP megasuites towards a more loosely coupled modular ERP environment. Despite the rigid nature of single vendor ERP suites, they provided for consistency and integrity of data and processes. In the modular ERP environment, the gain in flexibility comes at the price of needing to integrate the data among the modules. Gartner concludes that failing to address such issues could quickly erode the value users expect to get from the postmodern ERP.

In [5], Shen, et al. describe how computer supported collaborative design enables product designers to interact with a wide range of other designers of the product lifecycle. But although interaction is possible, the authors find that actual collaboration is often hindered because of a lack of uniform interpretation of product and process modeling languages and terminologies.
In [2], Frechette concludes that the biggest barrier to utilizing model data effectively throughout the enterprise is moving model data between the various engineering and business applications that make up the enterprise. The solution he proposes is a data format that is independent of the software applications used by the enterprise.

The CHAMP project was motivated by observations such as these to produce a data curation pipeline that would resolve semantic heterogeneity in the enterprise. The foundational component of this pipeline is a set of tiered ontologies that can be used to semantically model the products and processes across the product lifecycle. In addition to developing ontologies, the CHAMP project used, explored, and developed data management tools including the Ontological Semantic Concept Alignment and Refinement (OSCAR), natural language processing (NLP), optical character resolution (OCR), Process Workflow and Ontoview. In combination, these tools provide a prototype capability to ingest structured data into the semantically consistent ontological model and build reports based upon the aligned data.

In the remaining sections, we describe the components of CHAMP in the following order:

- Ontologies
- OSCAR
- NLP and OCR
- Process Workflow
- Ontoview

2 CHAMP Ontologies

2.1 Constraints

We begin by describing the set of tools and the modeling methodology used to develop the CHAMP ontologies. We classify these as constraints, since they are external to the design and development processes, yet they limit those processes in significant ways.

2.1.1 Web Ontology Language (OWL)

We follow the W3C recommendation that ontologies be published for exchange as OWL files (more precisely: as files using the OWL 2 Web Ontology Language). However, our conformance with the W3C recommendation regarding the use of OWL is not an endorsement of that language as being ideally suited to the task of publishing ontologies. Rather, it is a recognition that OWL is the current standard and the ecosystem of tools surrounding it makes it the current best choice. As technology evolves, other languages may replace OWL's role as the standard. For example, ISO has sanctioned an additional language, Common Logic (CL) (http://www.iso.org/iso/catalogue_detail.htm?csnumber=39175), and its use for ontology formulation is increasing (http://stl.mie.utoronto.ca/colore/ontologies.html). CL has greater expressive power than OWL, but lacks certain computational features and tooling that make it less attractive than OWL.

Similarly, work in the research field of ontology, such as the specification of Basic Formal Ontology (see below), is often written in first-order logic, but first-order logic lacks decidability—a feature of logical systems that is required for reasoners to detect violations of consistency. Because OWL is a decidable fragment of first-order logic, we take it as best practice that the canonical statement of an ontology be made in first-order logic, and its implementation in a particular language, such as OWL, be made conformant with the canonical representation, such that nothing said in OWL is in violation of what is said in the canonical representation in first-order logic. In this way, we attempt to balance the need to be formally exact in our expression with the practical need to implement ontologies that may be used according to adopted standards.

2.1.2 Basic Formal Ontology as Top-Level Ontology

Ontologies are classified according to level: top-level, mid-level, and domain-level. The lowest level ontologies are called 'application ontologies'.

The level of an ontology is determined by the level of generality of the types in reality that it represents. Object, for example, is a very general type and so the class 'Object' should belong to a top-level ontology; Living Thing and Person are less general types that belong to a mid-level ontology, since they are common across many but not all domains. Still less common types, such as Blue Eye Color, belong to a lower-level ontology, such as a domain ontology (i.e. the domain of eye colors). At the lowest level, application ontologies are built that service the needs of a particular application. For instance, an application may require a class 'Blue Eye Color with Shade RGB Value Range .20-.75'. Such a class would be particular to the needs of the application that requires the gathering of data according to the parameters of the class. Thus, a

class like ‘Blue Eye color with Shade RGB Value Range .20-.75’ would belong to an application ontology.

As will be explained more fully in the section on design principles below, we recommend a tiered architecture of ontologies starting with a single top-level ontology that forms the base of all other ontologies. The purpose of the top-level ontology is to provide a high-level, domain-neutral representation of distinctions such as that between objects and events, and between objects and their attributes (e.g. qualities and roles).

There are many benefits provided by a top-level ontology. One of the main benefits is that a top-level ontology’s high-level distinctions impose a certain amount of control on those ontologies that descend it, requiring these ontologies to continue to use their vocabularies in a controlled and precise manner. This means that if a top-level ontology distinguishes Processes from Objects, that a domain ontology that has a class such as ‘Person’, which it asserts to be a subclass of Object, cannot at a later time treat instances of processes as instances of ‘Person’; trying to do so would create inconsistencies that could be detected with the use of ontology reasoners, such as FaCT++ and Hermit. In such a way, use of top-level ontologies prohibits a tendency at semantic drift that leads to a lack of interoperability over the lifetime of an ERP system.

We recommend Basic Formal Ontology 2.0 (BFO) as the top-level ontology [1]. BFO is a small, highly abstract, top-level ontology designed for use ‘under the hood’. Its role is to provide a framework that can serve as a common starting point for representing types that are more specific in order to ensure consistent ontology development at lower levels in a way that maximizes the degree of interoperability among ontologies developed by different collaborating groups.

One reason for the success of BFO is that it is a strict top-level ontology. As such, it is domain neutral and does not contain its own representations of physical, chemical, biological, psychological, social, or other types of entities that would properly fall within mid- or lower-level domains. BFO is correspondingly very small, with a narrowly focused task: that of providing a top-level ontology that supports the integration of multiple heterogeneous domain ontology plug and play modules at lower levels.

2.1.3 Common Core Ontologies as Mid-Level Ontologies

The purpose of the Common Core Ontologies (CCO) is to extend BFO to a structured vocabulary that can be easily extended to represent the content of any specific data source whatsoever. The design of CCO is such that explicit connections between classes are not typically asserted. For example, the CCO class ‘Person’ does not have a number of axioms that relate it to other classes, such as ‘Weight’ or ‘Occupation’, in a way that would be analogous to the way attributes of entities are linked in a logical model of a relational database. The reason for this is that a guiding principle in the development and application of the CCO is to produce a vocabulary that can integrate all information from any data source about every type of entity, and not to prescribe, and thus limit, the types of information that should be collected or queried about a particular type of entity. To provide this functionality, the CCO defines a modular set of

extensible classes and relations that can be connected as needed at the instance level to translate information from data sources.

The Common Core Ontologies comprise a suite consisting of twelve ontologies that have been designed to cover particular domains. These domains include: agents (organizations and persons), artifacts, events, geospatial, time, qualities, and information. Whatever content from a particular data source that is not found in the Common Core Ontologies is then added to domain ontologies that extend from those in the common core. Building ontologies in this manner reproduces patterns of expression across domains that reduce the amount of time and effort required by analysts, query writers, and algorithm developers to acclimate themselves and their products to different content. As new data sources are encountered, existing semantic content from already built ontologies is re-used to the extent possible. Building ontologies in this controlled manner greatly reduces the proliferation of ontologies, which in turn reduces the amount of time and effort required to find content and build mappings between content. Importantly, the method enables rapid development by creating a cycle of ever diminishing extent of new content that needs to be ontologized. Figure 1 provides a visualization of the method.

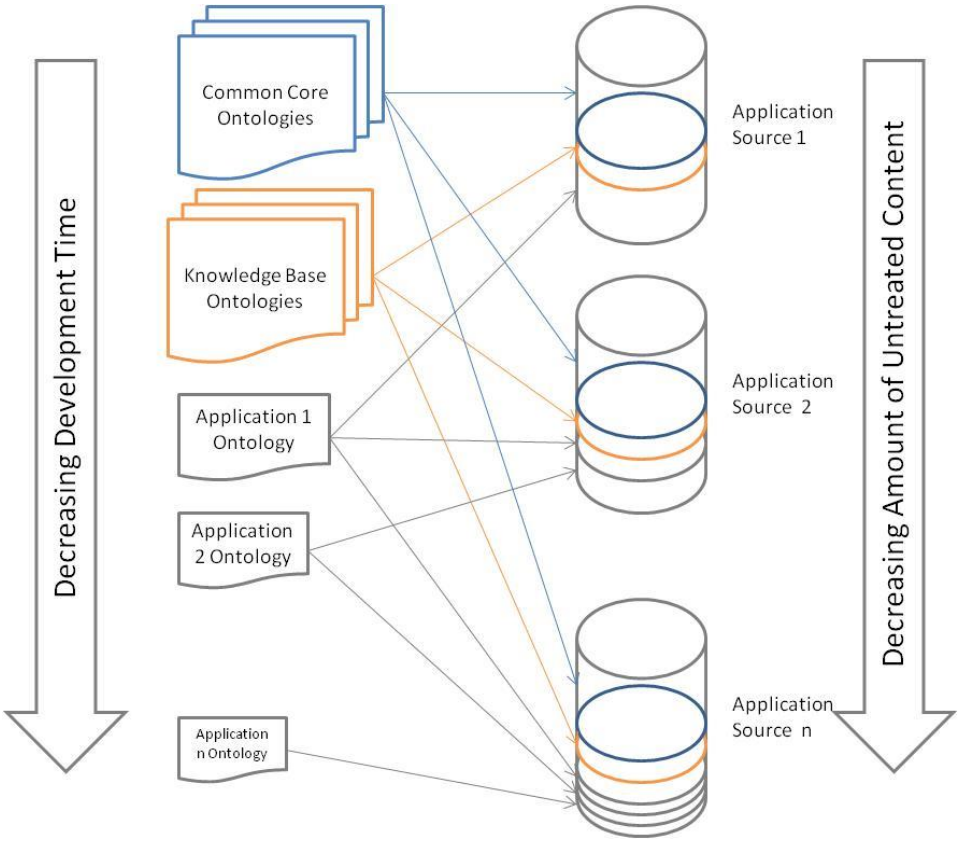

Figure 1 The Common Core development methodology

Each of the ontologies in the Common Core was developed alongside the others in the suite, allowing the files and their import ontologies to be used independently or in conjunction with one another. For this reason, the Common Core Ontologies have a modular design, and each ontology in the suite may be referred to an 'ontology module'. The CHAMP ontologies, in turn, extend from the Common Core ontologies and retain the modular design, such that each

CHAMP ontology module and its imported ontologies may be used independently of other CHAMP ontology modules.

However, the CHAMP ontologies, like those of the Common Core, were designed to retain their status as mid-level ontologies. This is because they were built to be reference ontologies that might impose conformity on ontologies that extend from them. Such ontologies include the lowest level ontologies: application ontologies. Application ontologies extend mid-level and top-level ontologies to provide terminologies that might service a particular application or purpose.

2.1.4 Ontological Realism

To maximize both utility and stability, the modeling process of ontology development should rest on what in [6] is called 'ontological realism', which amounts to the idea that an ontology should be analogous not to a data model, but rather to a reality model. Under this constraint, ontologies are representations not of the data to be integrated, but rather of the entities to which those data refer. Some ontologies will indeed need to contain terms referring to data items – for example, to types of images, or types of email, or of other text documents. By following the practice of ontological realism these ontologies will treat these data items as entities in reality in their own right.

Employing the methodology of ontological realism results in ontologies that are the best candidates for serving as common models of all the data sources within a complex information ecosystem. Realist ontologies serve as a proxy for some portion of the world, employing, as far as possible, consensus expressions from natural language, including scientific vocabularies. Data sources are descriptions of the world, but they also add a layer of perspective in order to serve the application specific needs of their users. Realist ontologies enable enterprise-wide data integration by circumventing this layer of perspective from each of the several data sources involved and thereby arrive at a depiction of the domain that can serve as benchmark for their integration.

Adopting the method of ontological realism also contributes to improved governance. In order for an ontology to adhere to the realist approach, all of its assertions must be true of the world; if they are not, they must be corrected in order to be so. Using the world as a basis for determining the correctness of an ontology provides a clear methodology for adjudicating disputes and continually improving an ontology's content; this is an important benefit for long-term management.

2.2 Requirements

We have now described four constraints: the use of OWL, the use BFO as a top-level ontology, the use of the CCO a set of mid-level ontologies, and the use of method of ontological realism. These four constraints provide the basis for building high-quality ontologies. In this section, we describe the requirements of that the CHAMP ontologies fulfill.

2.2.1 Consistency

An ontology is consistent if and only if every one of the classes denoted by its terms can have instances. In its simplest form, an inconsistent ontology would define a class such that any instances would need to have a set of incompatible characteristics, X and not-X. Of course, inconsistent ontologies are not built intentionally. More often than not, they are the result of oversights caused by the complexity inherent in formalizing some domain. As the number of terms, relationship expressions, and axioms within an ontology grows, so too does the likelihood of introducing an inconsistency. This likelihood is increased further as the number of ontologies within an enterprise grows, as ontology developers are provided with the need, and the opportunity, to import terms from multiple sources.

To test an ontology for consistency, there are a number of automated reasoners available that can trace through all of the relationships in the ontology and detect any inconsistencies. The majority of these reasoners are capable of performing consistency tests on ontologies written in the Web Ontology Language (OWL) DL dialect of OWL (OWL 2 DL).

2.2.2 Principle of Single Inheritance

The principle of single inheritance prescribes that each ontology class should be a subclass of one and only one ontology class. The resultant chain of single inheritance forms what is called the asserted taxonomy. Adherence to this principle ensures that everything that holds of a parent term holds also of (is inherited by) all descendant terms at lower levels. This means that each asserted taxonomy has a single root node and that every ontology will contain one or more asserted taxonomies as proper parts. All non-root terms will have exactly one parent, from which it descends via a *subclass of* or *is_a* relation.

This principle does not prohibit the use of subclass or equivalent class axioms whereby a class is defined to be a subclass or equivalent class of an anonymous class. The OWL language provides the means to express facts about entities using relations to other entities at the instance-level. For example, a fact such as “Automobile has part some Engine” uses the *has_part* relation to relate instances of automobiles to at least one instance of engine. The semantics of this fact is that Automobile has been made a subclass of the anonymous class “thing having at least one engine as part”. The class is anonymous because it is not part of the asserted taxonomy. The set of subclass and equivalent class axioms forms the *inferred taxonomy* of an ontology. In this inferred taxonomy, classes previously anonymous become explicit and the principle of single inheritance is no longer in force.

2.2.3 Definitions

Every class in an ontology (other than the root) should be provided with an associated human-readable definition. Standardly, this association is achieved by adding annotation properties, which are a means for associating metadata with elements in an ontology file. Whereas the use of the RDFS annotation property ‘comment’ is acceptable, it is preferable that a new annotation property be created expressly for this purpose.

The human-readable definitions of classes in the ontology are required to be always of the genus-species form:

an *S* = $_{\text{Def.}}$ a *G* that *D*s

where 'S' (for: species) is the term to be defined, 'G' (for: genus) is the immediate parent term of 'S' in the relevant ontology, and 'D' (for: differentia) provides the species-criterion; that is, it specifies what it is about certain G's that makes them S's. As ontologies evolve, it is expected that the human readable definitions will be translated into subclass or equivalent class axioms as described above in the section on single inheritance.

Examples:

- Natural Event $=_{\text{def.}}$ a Process that is not caused by any human act.
- Geographic Event = $_{\text{def.}}$ a Natural Event that affects some Geographic feature
- Landslide $=_{\text{def.}}$ a Geographic Event in which there is a rapid descent of soil or rock down a mountainside.

As more specific terms are defined through the addition of ever more detailed differentiae, their definitions encapsulate the information regarding child-parent links connecting each class all the way back to the corresponding root. At the same time, the task of formulating definitions serves as a check on the correctness of the constituent hierarchies in these ontologies.

Use of the genus-species definition structure helps to ensure that definitions are not circular (when the term defined appears in its own definition), and thus that they communicate information that is of value to the user – in conformity with the principle that a definition should use only terms that are easier to understand than the term defined.

Definitions are required also for (non-root) object properties, and these two should as far as possible be defined using the genus-species rule.

2.2.4 Preservation of Meaning of Higher-Level Ontology Terms

It is a fundamental principle of best practice that one re-use terms from other ontologies, since re-use contributes to the interoperability of ontology modules and thus also of the information systems that these modules support. However, if not performed carefully, the reuse of terms can bring the risk of altering the meaning of the re-used term. The most common way in which this type of problem occurs is when an ontology reuses a term from a higher-level ontology, but adds to its content through the addition of an axiom. An example is reusing the term Organization but adding an axiom that asserts that every such organization has exactly one leader. To be conformant, the creator of the lower-level ontology should either request that the curators of the top-level ontology add the axiom, or introduce into the lower-level ontology a subtype of Organization (e.g. Single Leader Organization) to which the axiom could then be added without altering the meaning of the original.

2.3 Accomplishments

2.3.1 The Industry Ontology Foundry

Despite its early termination by DMDII, the CHAMP project has succeeded in having a significant impact on industry work in ways both large and small. In the Spring of 2017, Clare Paul at the Air Force Research Laboratory and Hedi Karray at the University of Toulouse were invited to the University of Buffalo for workshops dedicated to representing the domain of materials. These discussions resulted in a re-orientation of their work, which now uses Basic Formal Ontology as a top-level ontology, along with our series of design principles and methods drawn from the BFO framework. The researchers at the University of Buffalo are grateful that CHAMP has supported these partnerships.

However, the greatest impact of the CHAMP project on industry will be to seed the Industry Ontology Foundry—an initiative modeled on the Open Biomedical Ontology Foundry [7]—whose purpose is to create a large, expert-curated suite of interoperable high-quality ontologies covering the domain of industrial (especially manufacturing) engineering. These ontologies will be developed in tandem to ensure that they promote interoperability, and each will be small but easily extensible for different companies concerned with particular domains. In parallel to the CHAMP project, planning discussions have been occurring among individuals at a number of different organizations.

Organizations participating in the Industry Ontology Foundry presently include:

- NIST
- Air Force Research Laboratory
- Airbus
- Autodesk
- Cambridge Semantics
- CIMData
- CUBRC
- Dassault Industrie

Following the present discussions among IOF participants, the IOF will provide an online repository for vetted and curated ontologies of industry. Each of these ontologies will be committed to a series of shared design principles, whose purpose is to both raise the quality of the ontologies and to ensure a greater degree of interoperability and thereby to encourage re-use. The CHAMP project was conceived as a stepping stone in achieving this vision, with its mid-level reference ontologies to be used as drafts that will undergo refinement among members of the IOF. The software produced by CUBRC as a result of CHAMP will also be disseminated among IOF members and its integration encouraged within the future IOF platform.

2.3.2 Education

The CHAMP project has also made substantial contributions to engineering education by providing funding for three students attached to the CHAMP project, with an additional three engineering students attached to the project as well. These six graduate students, five of whom are from the engineering school at the University of Buffalo, gained experience: building ontologies in OWL using the ontology editor Protégé; mapping tabular data structures to OWL

ontologies to create RDF triple stores; and doing so within the domain of the product life cycle. This experience has already opened up opportunities for some of them to continue this work by collaborating with other researchers elsewhere on related projects.

In addition to the opportunities offered to students, CHAMP has led to collaboration among an international network of academic partners, who are now active collaborators on the IOF.

These institutions and their members include:

- INP-ENIT, University of Toulouse (Hedi Karray)
- Clemson University (Venkat Krovi)
- École polytechnique fédérale de Lausanne (Dimitris Kiritsis)
- Loughborough University, UK (Bob Young)
- National Center for Ontological Research (Kemper Lewis, Rahul Rai, and Barry Smith)
- Penn State (Timothy Simpson)
- Texas State (Farhad Ameri)
- UMass Amherst (Ian Grosse)
- University of Toronto (Michael Grüninger)

2.3.3 The Product Life Cycle Ontologies

As we began to plan out the ontologies we were to design, we took as our domain the product-life cycle, from the initial product design phase through production, testing, use, and the end-of-life phase. To approach a domain of such significant scope, we began by working in the constraints described above, re-using other ontologies (namely, BFO and those of the CCO) and conceiving of more general, mid-level ontologies that may be extended in other ontologies under development, or by others for particular purposes in application ontologies. This was carried out in order to complete the goal, specified in our contract, to: “construct a semantic model establishing a manufacturing domain ontology (vocabulary and relationships) representative of the prevalent concepts and associations that are common across small to mid-sized manufacturing companies holistically.”

Our representations of the domain of product-life cycle were constructed in dialogue with experts in industry and engineering at the University of Buffalo, the Air Force Research Laboratories, and NIST, as well as with our industry partner, Cobham. Cobham provided us with data from the domains of manufacturing and design, which were used to construct particular application ontologies to which we hoped to align data in order to create RDF triple-stores that could be load tested. These alignment and load testing tasks did not occur due to the early termination of the project. However, these application ontologies were also generalized in the creation of mid-level ontologies of design and manufacturing, such that they could be both useful and publically available. As a result of working both from expert-level descriptions of the domain, as well as from the vocabulary particular to Cobham, we were able to test our initial representations of the product life-cycle against the coverage necessary to service lower-level domain ontologies particular to the vocabulary of Cobham’s many data sources.

There are seven mid-level ontologies in the product-lifecycle ontology suite developed under CHAMP that are in a sufficient state of completion that they merit a public release. Stating clearly their state of completion would be impossible, given that there are no explicit standards for evaluating the readiness of ontologies according to which they could be rated. The seven include ontologies that cover the following domains: commercial entities (e.g. products and economic goods), design, maintenance, manufacturing processes, the product life cycle, testing processes, and tools. Each contains classes and object properties relevant to the representation of their domain, and each class has a definition conforming the genus-species form detailed above. Furthermore, the commercial entities ontology makes use of axioms in OWL to distinguish many of its defined classes, including Product, Economic Good, and Service, from the asserted taxonomy containing Material Entity, Software, and Intentional Action.

Each of these seven files constitutes a mid-level ontology that imports the whole of the CCO, as well as BFO, while continuing the modular approach of the CCO. The import structure among the files themselves can be seen in Figure 2 below. Each file imports the Common Ontologies and Basic Formal Ontology, but individual files vary; for example, with the Commercial Entities Ontology having no other imports, while the Maintenance Ontology imports both the Product Life Cycle Ontology and its import, the Commercial Entities Ontology.

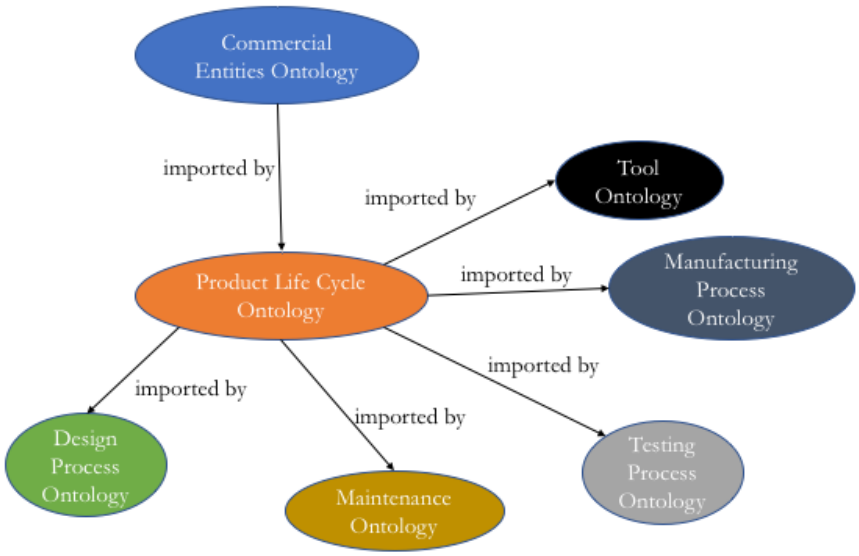

Figure 2 The import structure among the seven ontologies of the product life cycle suite

As part of this project, we had also been working on an ontology of Unix permissions, which was designed to facilitate the integration of access permissions in a system relying solely on an RDF back end. Second, we had also been aiding in the construction of an ontology of materials properties, aiding Clare Paul at the Air Force Research Laboratory in its development. This ontology covers material properties such as stress, hardness, as well as phase states of matter (e.g. liquid, plasma, and solid) and would have been integrated with the ontologies we had developed, and it would have become the parent ontology of two other ontologies we had

developed independently of the CHAMP project: an ontology of additive manufacturing and an ontology of functionally graded materials. Third, we had planned to also engineer an ontology concerning what we informally called 'business flow' that would have represented the domain of documentation, including processes of signatory approval, forms, templates, budgets, contract evaluation processes, and budgeting, all of which is vital to the representation of ownership, commerce, and planning. Such work would have re-used portions of the Document Acts Ontology—a publically available, open source ontology presently used in many projects in bioinformatics. Finally, we had planned to continue developing the ontologies that carried through to the end of the product life-cycle, including ontologies of recycling and disposal, the use of artifacts, and logistics. These projects will not be carried out under CHAMP as a result of the early termination decision, but we hope later efforts, including those of the IOF, will revive some of them.

The seven ontology files completed as part of CHAMP will continue to evolve following their public release as part of the IOF effort as they are critiqued and re-worked by others working in the field. In our review, we found no comparable set of modular ontologies developed for industry that are rooted in a principled approach to ontology design, and thus that share in our long-term vision of open-source ontology curation and re-use. The participant organizations in the IOF know this as well, and the need for such work remains a driving motivation behind their endeavors.

2.4 Lessons Learned

There are well-known problems in ontological engineering that were made more salient in working in the domain of industry. Here, we describe two that we encountered, with which any project adopting ontological realism will also contend.

First, the domain of materials requires the tracking of portions of material (e.g. a portion of laminate composite, a stack of steel beams, a volume of water in a tank) as they undergo change. Change in a portion of material can occur both as a result of molecular change (for instance, a change in its material phase state) or as a result of the gain or loss of a material part. Basic Formal Ontology provides resources for tracking such entities as they undergo change (see [3] for discussion), but even here we uncovered two limitations. The first of these concerns the issue of 'gappy' objects, as when a watch at time 1 is disassembled at time 2 and put back together at time 3. The issue of gappiness concerns whether or not we should affirm that portions of material, like the watch, endure through their dissolution or not; i.e. should an ontology affirm that it is the *same* watch at time 3 that it was at time 1 or a new watch at time 3, with the original watching permanently ceasing to exist at time 1 (similarly, is a portion of material that is divided, then brought together again, the same portion). Second, BFO is committed to a nonmultiplicativist philosophy according to which instances rigidly belong to asserted classes, such that an instance of Object is always an object, and if it ceases to be an object, it thereby ceases to exist. However, there is a significant need to acknowledge that materials are both objects at one level of description, while also object aggregates composed of molecules at another level of description. Given BFO's commitment to nonmultiplicativism, it remains an open question how best to account for the fact that objects appear to belong to distinct classes at different levels of description. This is an issue for which further experimentation is necessary.

Second, in our research, we encountered no ontology or taxonomy of artifacts (i.e. tools, instruments, machines, computers, etc) that seemed sufficiently motivated by principle to count as superior to every other ontology of taxonomy. In our review, most were either very poor, or were built around an excellent standard that was, nonetheless, not common across industry, or consisted of a mere list of artifact types whose classification seemed boldly open to critique. In the future, the domain of artifacts requires special attention, since a taxonomy of artifacts and a taxonomy of artifact functions is often doing double-duty within an ontology (representing the same domain in overlapping ways) when one taxonomy could do both. In addition, other features of artifacts, such as the domain in which an artifact typically operates (e.g. kitchen tools, carpentry tools, injection molding tools), the type of process in which the artifact is typically employed (e.g. screwing, molding, sanding), or the certification necessary for the prescribed operation of the tool (e.g. fork-lift license) are, in addition to many others, alternative ways of querying a tool catalog that are over and above the work performed by a taxonomy of artifacts alone.

Although these issues occur across domains, they are somewhat more pressing in industry, which is particularly concerned with artifacts and materials. Thus, they reveal deeper problems with ontologizing data across the product life cycle than has been appreciated elsewhere by those tasked with developing modeling standards that may serve the end of data integration. Had we time, this section may have articulated steps toward practically addressing these issues in our implementation, but due to our earlier termination, we only articulate them here in the hopes that others may see them clearly and address them forthrightly.

3 Working with Data: Alignment and Ingestion

A primary concern of the CHAMP project was making semantic technologies practical to implement. If a company is to make day-to-day use of our ontologies to create, store, and interact with data in the form of RDF triples, then at least two general capabilities are required.

First, a company must have a way of aligning the future data it acquires, as well as its existing data, to our ontologies in order to produce RDF triples, so that queries may be run across their data. We suspect most companies are not so different from Cobham in having data that appeared in the following forms: relational databases (SQL), tabular data in the form of Excel files, specifications and reports in the form of unstructured documents containing tables and other elements (e.g. Word documents), CAD files stored in a variety of formats, and image formats (e.g. TIFF).

For many mid-sized companies, the majority of their data is likely stored in a relational database, but for many day-to-day activities, employees likely continue to work with a windows-folder hierarchy accessed via series of hyperlinks, where key documents were stored that both reported on and prescribed various work processes. CHAMP was well-prepared to deal with relational databases, as there are already many tools available for linking relational databases to ontologies. CUBRC's in-house application, OSCAR, provides enhancements to the open source application 'KARMA'—a program used for aligning tabular data sources to ontologies to produce RDF. However, in order to accommodate data in other formats, other tools were needed—in particular, what was required was an alignment tool that could save and call back RDF triples to be used in further alignment tasks. For this task, we created the application 'Process Workflow', which we describe below.

Second, employees of the company must have the ability to browse and query their data without being experts in semantic technologies. In particular, the query language for semantic web, SPARQL, has a steep learning curve, and would not be practical for day-to-day users to learn and work in. For this reason, we created 'OntoView', a technology that allows users to browse and query data by providing a GUI on top of a SPARQL engine.

These two technologies are prototypes, but they address—like no other technology with which we are familiar—the concern of how companies are to work and interact with data stored in RDF. Because CHAMP is interested in providing a very low-cost option for increasing interoperability for companies, we are excited by the direction of these technologies.

In an effort to explore how granular we could get in our data alignment, we also explored NLP as a solution to extracting data from unstructured text (e.g. Word documents). These results are also reported below.

3.1 OntoView

3.1.1 Overview

OntoView is a technology that exposes data stored within a Resource Description Framework (RDF) triple store and provides a web-based graphical user interface (GUI) to view and query that data. Users build SPARQL Protocol and RDF Query Language (SPARQL) queries containing basic graph patterns derived from the traversal of relationships between International Resource Identifiers (IRIs), and alter that query using a guided syntax editor GUI. This technology leverages the CCO for resource identification and a taxonomy-based IRI search capability.

Knowledge of data schema is a key factor in building queries. When the data schema is sufficiently large, exploring data of interest is critical to understanding the data schema and is useful for refreshing the data analyst's mental model. OntoView provides a GUI that facilitates data exploration and query building through data exploration.

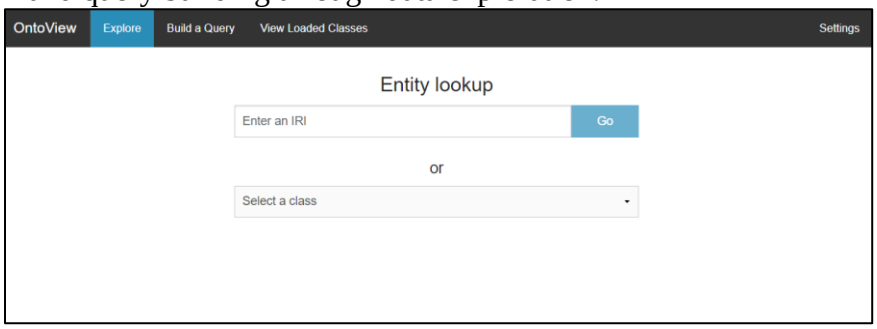

Figure 3: OntoView landing page

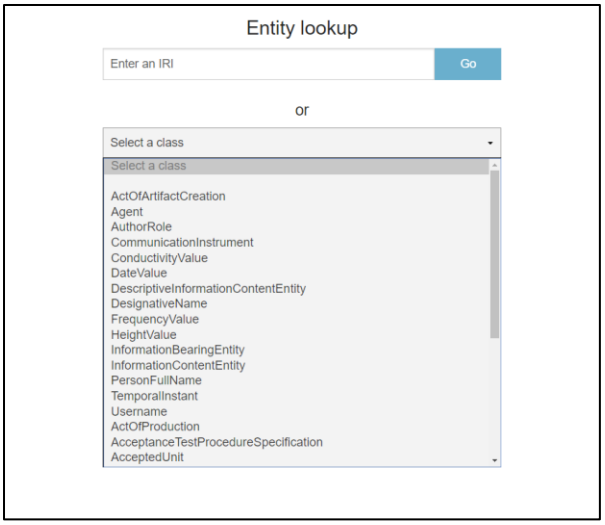

Figure 4: OntoView taxonomy search

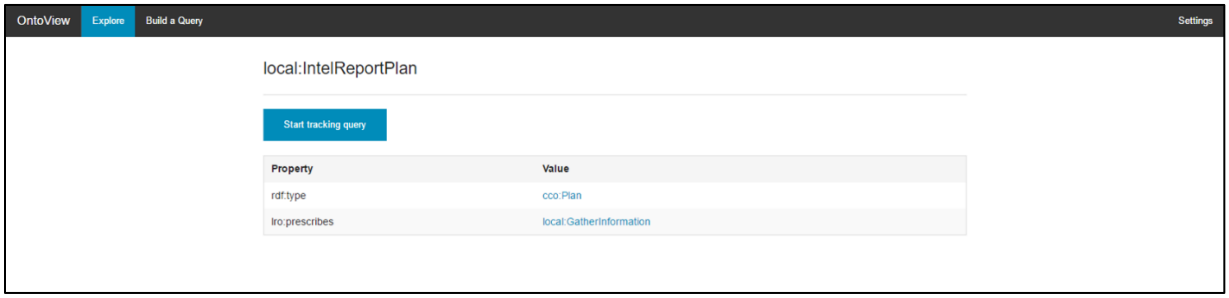

Figure 5: OntoView search result

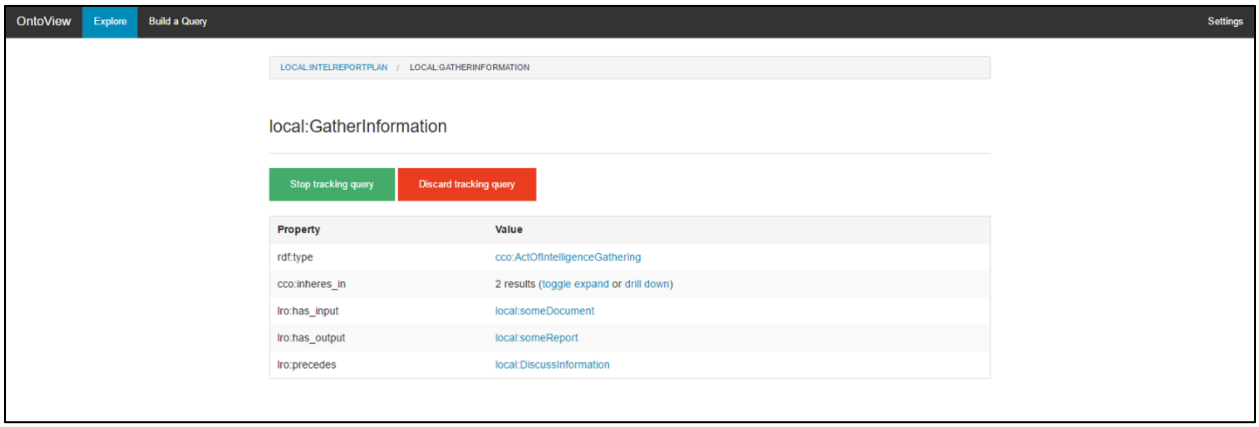

Figure 6: OntoView data exploration breadcrumb

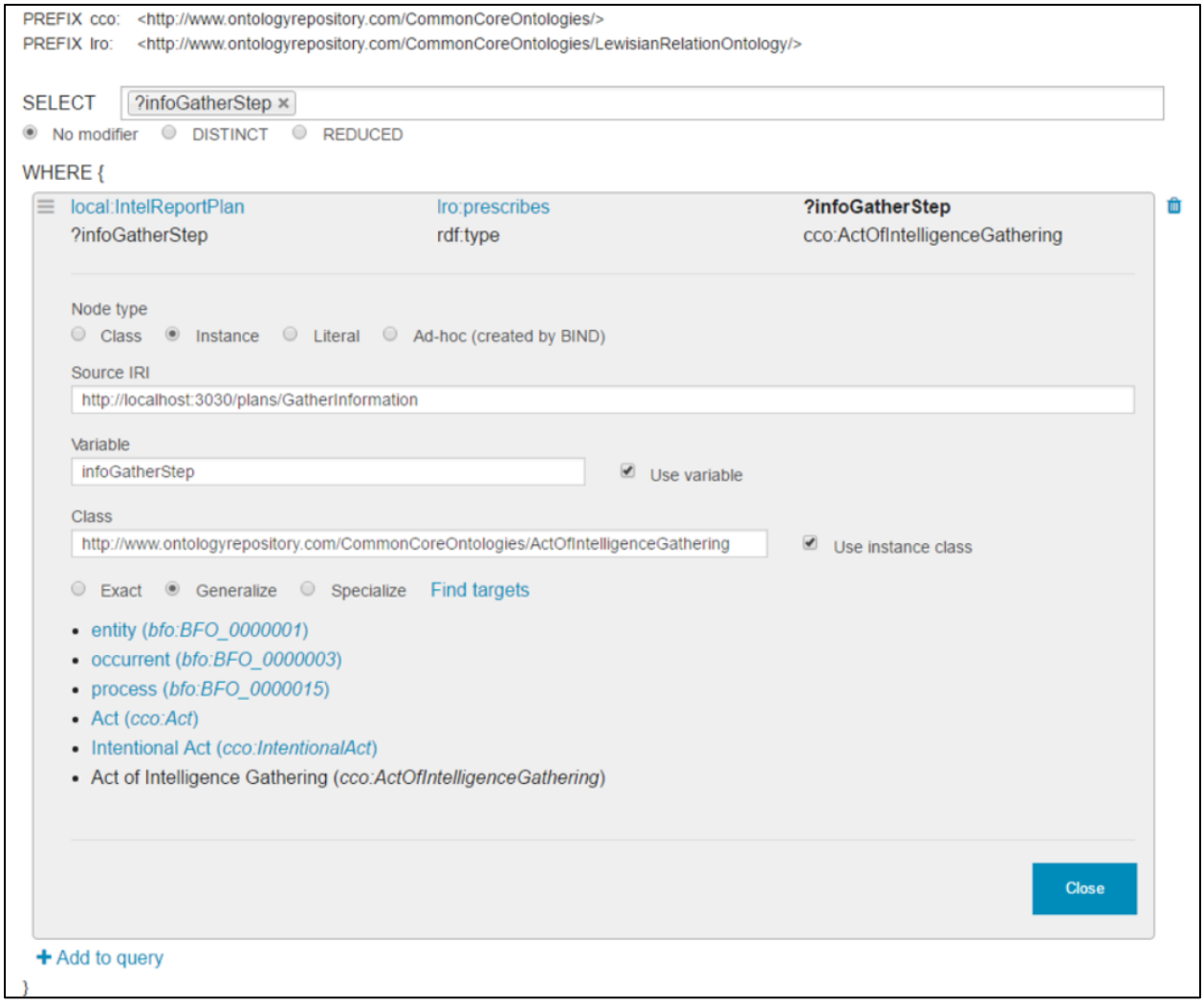

Figure 7: OntoView query builder

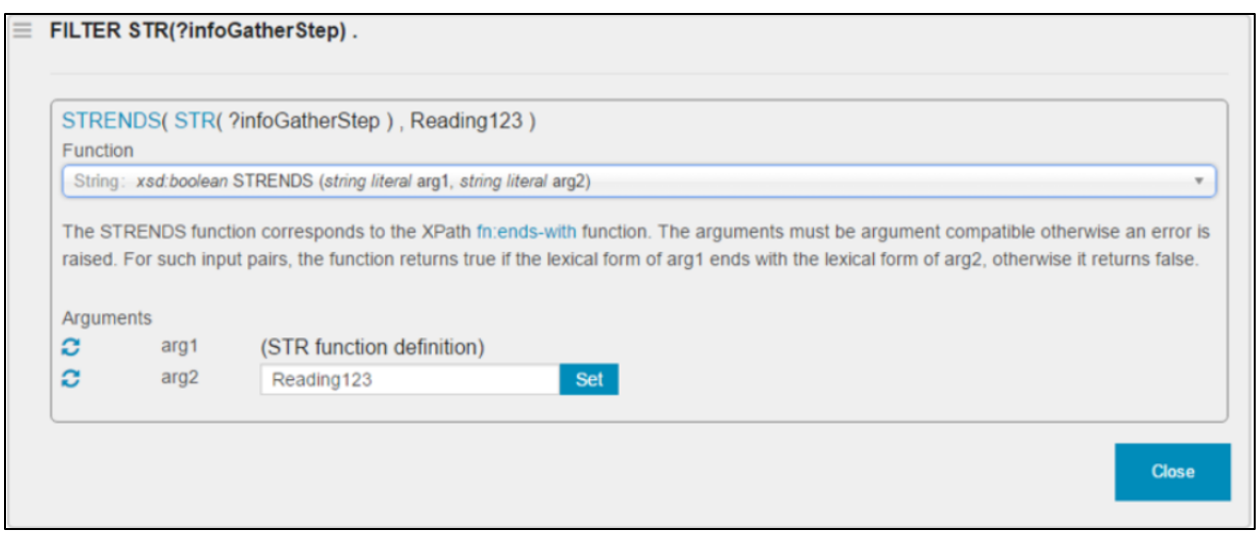

Figure 8: OntoView function editor

3.1.2 Requirements

OntoView fulfills the following high-level requirements:

- Provide "one-hop" navigation through data stored in RDF format
- Provide real-time configuration of triple store access parameters
- Meet the following CHAMP specific requirements:
    - Provide improvements on SPARQL query building:
        - Provide ad-hoc SPARQL query building capabilities
        - Provide complex SPARQL query clause building capabilities, such as FILTER, BIND, OPTIONAL, and UNION clauses
        - Provide query variable management
        - Provide SPARQL 1.1 function reference information
    - Provide taxonomy search through class exposure

3.1.3 Architecture

The OntoView architecture consists of a browser-based front-end GUI and a Java server backend. The front end communicates with an external REST-enabled triple store, with the requirement that the triple store communicates results in SPARQL 1.1 Query Results JSON Format. The back end primarily serves the endpoints that the front end displays, and also provides CRUD operations for front-end configuration. The following diagram illustrates the architecture:

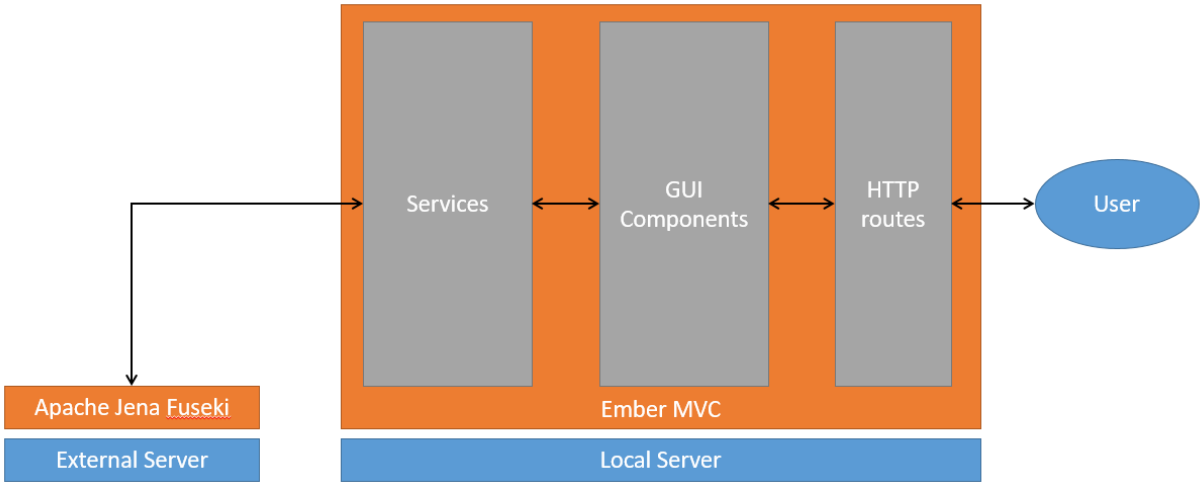

Figure 9: OntoView architecture

OntoView has the following library and system dependencies:

- Front end
    - Node and NPM: JavaScript build platform and package manager
    - Ember: client side MVC framework
    - Foundation: responsive front end CSS and JavaScript framework

    - Font-Awesome: font and CSS toolkit
    - ChosenJS: plugin providing functionality on top of select boxes
- Back end
    - Java 8
    - Maven: Java project management and build tool
    - Spring and Spring Boot: framework used for dependency injection and web servlet implementation
    - FasterXML Jackson: library used for JSON manipulation
    - SnakeYAML: library used to process YAML 1.1
    - Apache Jena: Semantic Web framework
    - Various Apache Commons projects
    - Jetty and JBoss Wildfly: enterprise application containers

The front end is built on top of the Ember MVC framework. Data primarily flows from an RDF triple store straight to the front end through an Ember service, rendered through GUI components. As the user navigates through data, another Ember service tracks progress through the dataset and passes that state to the query builder should the user choose to end progress tracking.

There exist four distinct user flows:

- Data browsing without progress tracking
- Data browsing with progress tracking, leading to a query built on the exploration breadcrumb
- Building ad hoc queries
- Editing OntoView settings configurations

All user flows are accessible by direct route by the user, or navigable by interface controls from the landing page.

3.1.4 Accomplishments

OntoView, as improved through CHAMP, replaced an earlier version of itself by adding usability improvements and the features listed above.

OntoView allowed for faster turnaround of Process Workflow (see below) feature development. The data transparency provided by both the taxonomy search and data browsing was instrumental in verifying instance data created by Process Workflow.

3.1.5 Lessons Learned

“One-hop” data browsing is good for exploration, but requires many clicks to get to multiple end results, i.e. a collection of literals for an entity. An entity view based browsing solution was designed and partially implemented, but there wasn’t sufficient time to complete nor demonstrate the capability.

The query builder interface was designed for an intermediate to advanced user; novice users would have built queries using the entity view based browsing solution mentioned above.

3.2 Process Workflow

3.2.1 Overview

Process Workflow is a collection of prototypical technologies that provide manufacturing process creation, storage, retrieval, and display with GUI. The software leverages process, product life cycle, tool, and design ontologies, as well as the CCO for common representation. Process Workflow provides a prototypical GUI for creating processes using the CCO as the common format, as well as a translation layer providing CRUD operations for process records.

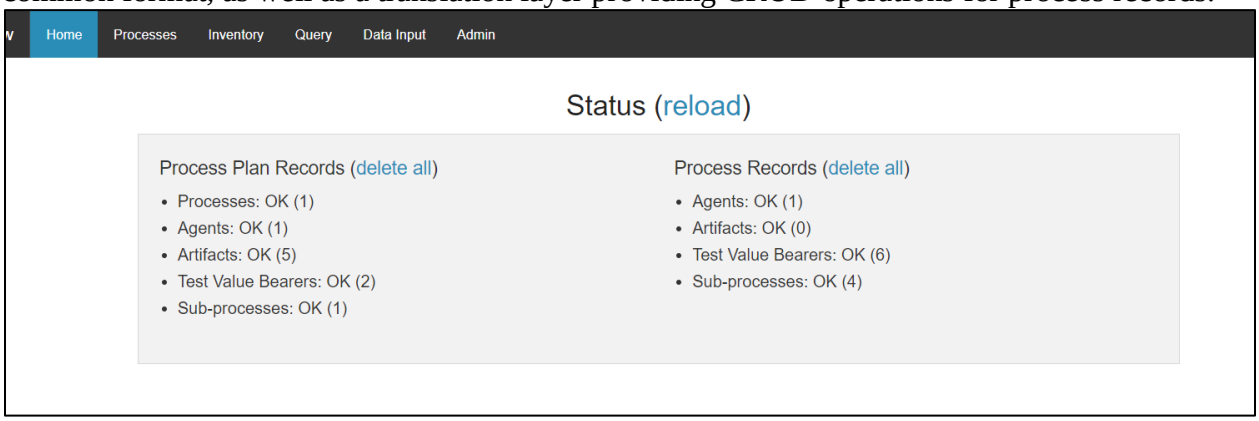

Figure 10: Process Workflow landing page

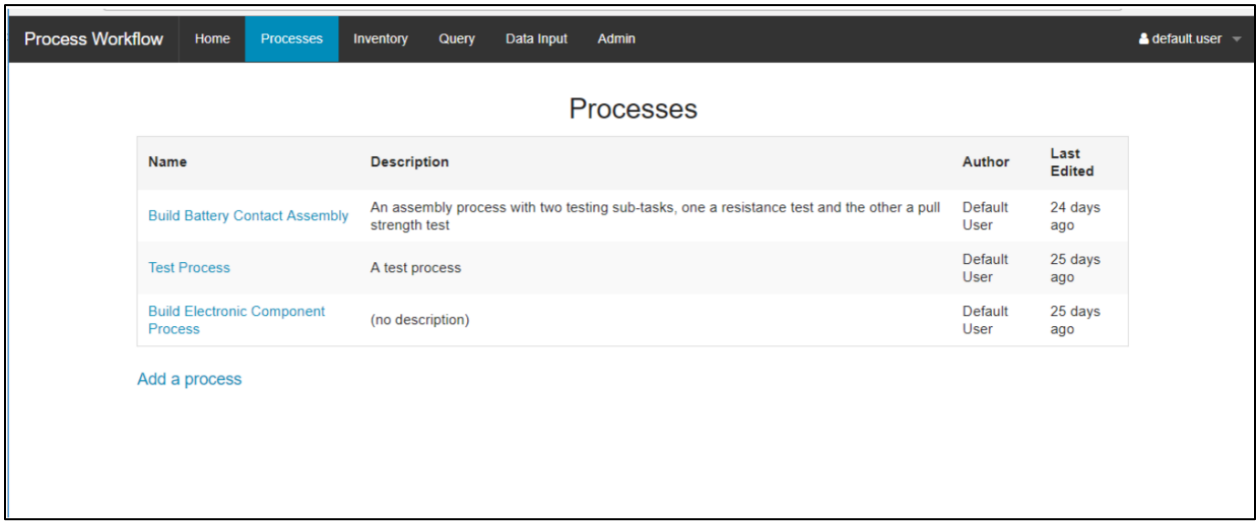

Figure 11: Process Workflow processes table

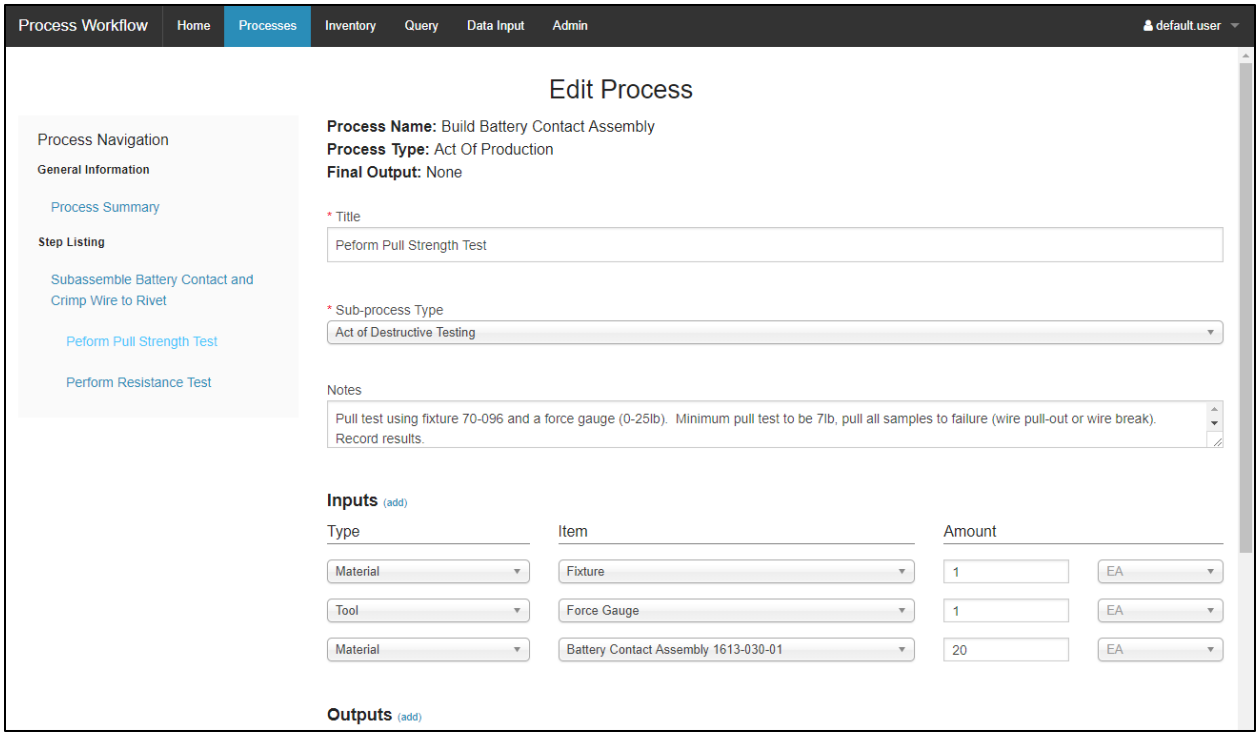

Figure 12: Process Workflow process editor

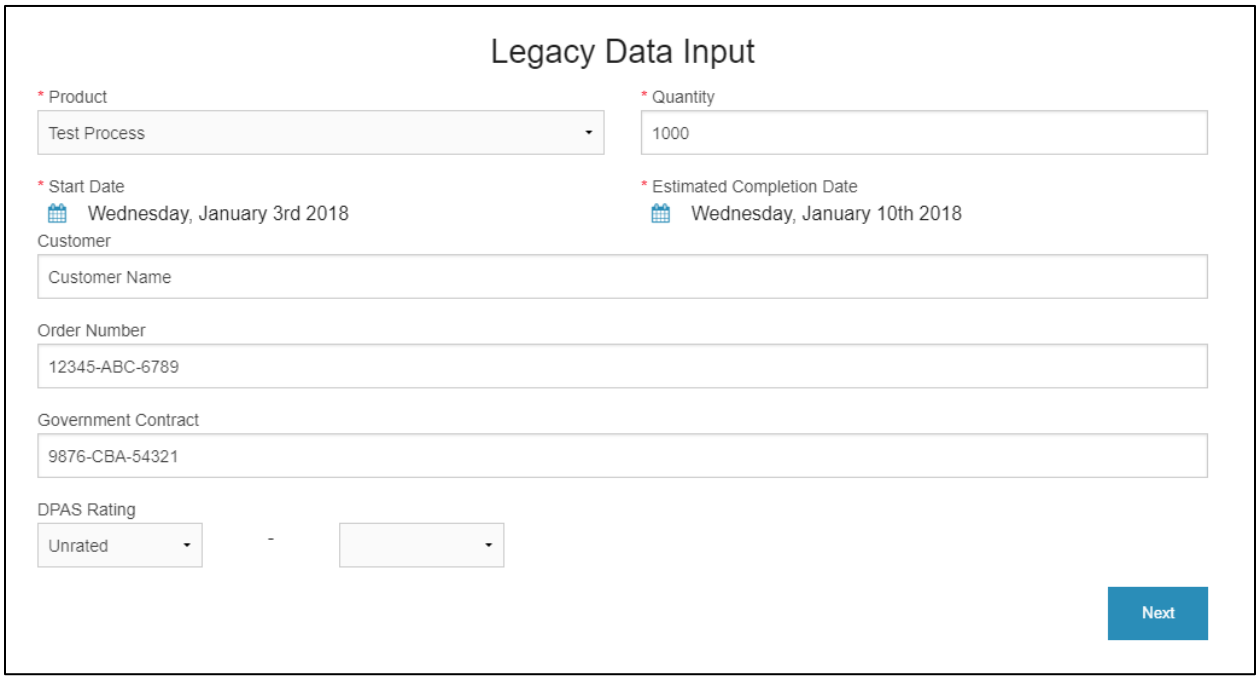

Figure 13: Process Workflow legacy data input landing page

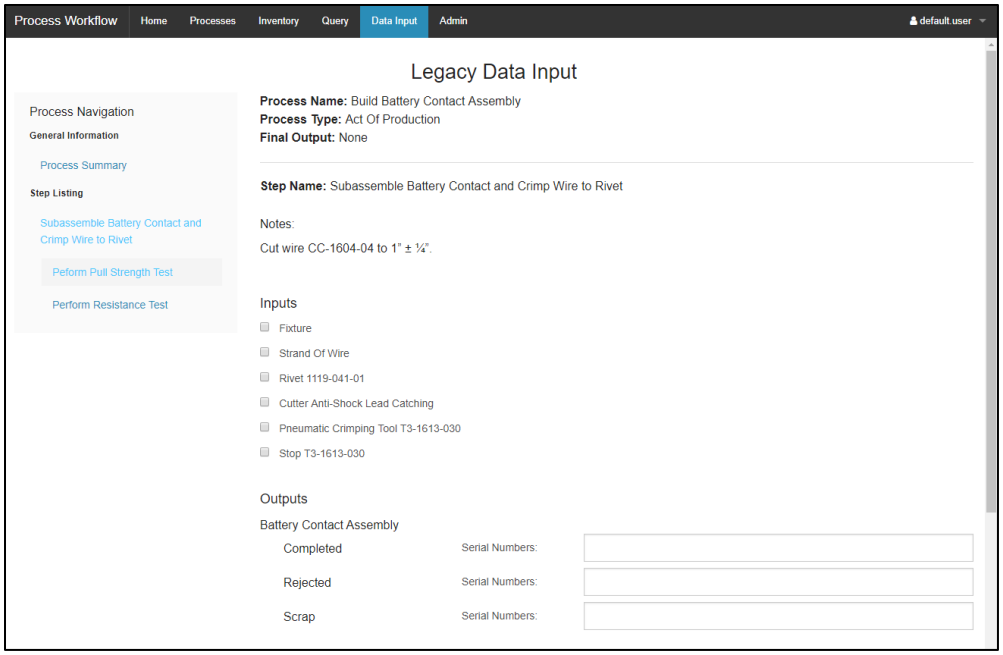

Figure 14: Process Workflow legacy data input

3.2.2 Requirements

Process Workflow fulfills the following high-level requirements:

- Provide a user workflow to create manufacturing processes using CCO semantics
- Provide legacy data input for alignment of existing realized process data to the CCO

3.2.3 Architecture

The Process Workflow architecture consists of a browser-based GUI backed by a Java server responsible for data persistence and serving endpoints that the displays. The following diagram illustrates the architecture:

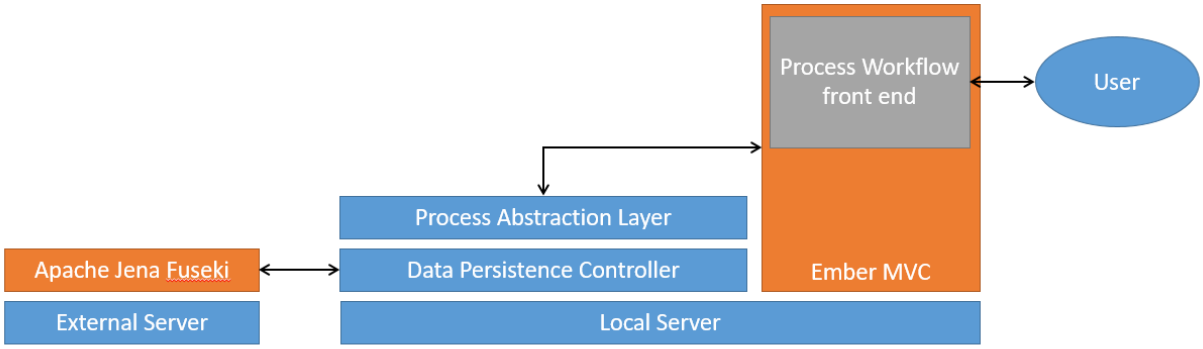

Figure 15: Process Workflow architecture

Data flows primarily from an external triple store through the Java server back end where it is condensed to process records, then flows out as JSON-API to the client front end. The client consumes the JSON-API through data adapters at the application level and converts it to Ember model records for display by Ember components.

Process Workflow has the following library and system dependencies:

- Front end
  - Node and NPM: JavaScript build platform and package manager
  - Ember: client side MVC framework
  - Foundation: responsive front end CSS and JavaScript framework
  - Font-Awesome: font and CSS toolkit
  - ChosenJS: plugin providing functionality on top of select boxes
  - MomentJS: library used for parsing, validating, manipulating, and displaying datetimes
  - Pikaday: plugin providing calendar picker functionality and GUI elements
- Back end
  - Java 8
  - Maven: Java project management and build tool
  - Spring and Spring Boot: framework used for dependency injection and web servlet implementation
  - FasterXML Jackson: library used for JSON manipulation
  - SnakeYAML: library used to process YAML 1.1
  - Apache Jena: Semantic Web framework
  - Various Apache Commons projects
  - Jetty and JBoss Wildfly: enterprise application containers

There are three distinct user flows:

- Creating a new process
- Editing an existing process
- Entering legacy data

All user flows are accessible by direct route by the user, or navigable by interface controls from the landing page.

3.2.4 Accomplishments

Process Workflow provided a representative data translation layer between objects used by the front end client and the CCO common representation. This is useful and applicable outside of the current usage.

3.2.5 Lessons Learned

Early iterations of the software followed loosely-defined user workflows; this caused some undue constraints on the user while creating processes. These were mitigated through usability improvements and interface transparency into available data records.

The original requirements specified prototypical GUI components for creating and generating data reports; insufficient development time was available to include these features.

3.3 Natural Language Processing

3.3.1 Overview

'Natural language processing' (NLP) refers to a collection of technologies required to parse unstructured and natural language documents for the purposes of extracting and exploiting information from those documents. The extracted information typically falls into four areas: events, entities, locations, and dates. NLP is a fairly mature technology that is founded on the known fundaments of language structures (e.g., English sentence canon is subject-predicate-object), orthographic principles (e.g., conventional rules of written text, cf. spoken dialogue), and semantic compositionality (e.g., figurative and literal uses of language).

'NLP pipeline' refers to the chained sequence of analytics through which data progresses during an extraction task. A common first step in an NLP pipeline is *tokenization*, which is the identification of each individual *token* in the text. At a basic level, a token can be thought of as a word, but tokens are also punctuation marks, abbreviations, numerals, units of measurement, etc., which are not typically considered words in the strictest sense. A standard pipeline, after tokenizing a document, is then able to identify or make conjectures about sentence boundaries, assign part of speech attributes to each token, identify syntactic constituency, assign semantic roles to the arguments of verbal phrases, and finally present the semantic proposition(s) of each sentence, in whatever format has been specified.

As a part of CHAMP, we explored the use of NLP for ingesting documents whose contexts consist largely of specifications frequently used in standards and work instructions. Our research was carried out on a work instructions document provided by Cobham.

3.3.2 Lessons Learned

For a full account of the lessons learned, see a full report included in our documentation, where we describe our experiment process and provide examples from the Work Instructions provided by Cobham using the JET NLP engine. The basic finding was that NLP did not work well on the work instructions because the instructions cannot reasonably be considered natural language. Rather than having the 'subject-predicate-object' form of natural language sentences, many instructions lacked a subject (e.g. 'Apply lubricant to the gear shaft') and contained nested or bulleted lists that represented an order of operations to be executed by a worker. These elements, together with all the part numbers and references throughout, made these documents poor candidates for NLP.

Although it would have been nice to have easily extracted specifications from these documents using NLP, for Cobham engineers, this would have been an over delivery. From their perspective, merely finding the documents that specify standards and other information relevant to a planned manufacturing process is itself a win, and our ontologies already make this possible. This is done by representing the document as an instance of Specification, and relating it at the instance level to an instance of Process (e.g. a Manufacturing Process). This link already facilitates the representation of industrial processes and the documents that contain standards, instructions, and other prescriptions that guide these processes.

Because NLP was not a fruitful approach to the extraction of data from unstructured documents, where such extraction is valuable, we suggest using a solution such as M-Turk, and paying workers a fee to paste unstructured document elements into a template whose columns are already mapped to terms in the ontology. For companies seeking a low-cost solution to this highly granular level of data extraction, this may be a quick, valuable investment that has the potential to replace large manuals containing standards and instructions.

4 References

[1] Arp, R., Smith, B. and Spear, A., 2015. Building Ontologies with Basic Formal Ontologies.
[2] Frechette, S.P., 2011. Model based enterprise for manufacturing. Duffie, ed., Omnipress, Madison, WI
[3] Otte, J. N., Ruttenberg. A. "BFO: Basic Formal Ontology". *Applied Ontology* [Forthcoming]
[4] Pettey, C, 2016, 5 Ugly Truths About Postmodern ERP, https://www.gartner.com/smarterwithgartner/5-ugly-truths-about-postmodern-erp/
[5] Shen, W, et al, 2008, Computer supported collaborative design: Retrospective and perspective, Computers in Industry, 59/9: 855-862.
[6] Smith, B. and Ceusters, W., 2010. Ontological realism: A methodology for coordinated evolution of scientific ontologies. *Applied ontology*, *5*(3-4), pp.139-188.
[7] Smith, Barry et al. "The OBO Foundry: Coordinated Evolution of Ontologies to Support Biomedical Data Integration." *Nature biotechnology* 25.11 (2007): 1251. *PMC*. Web. 10 Jan. 2018.