

Ontology for the Semantic Enhancement, Database Definition and Management and Revision Control

Edward S. Blurock ^a

Blurock Consulting AB, Lund, Sweden

Keywords: Ontology Use Case, Database System, Chemical Kinetics, Experimental Data, Chemical Modelling.

Abstract: This paper describes the use of ontologies interacting with a noSQL database (Google Cloud Firestore) in multiple capacities in the database system CHEMCONNECT. The motivation is to implement the ‘Data on the Web Best Practices’ as recommended by the W3C (<https://www.w3.org/TR/2017/REC-dwbp-20170131/>, 2017) in an application within the physical chemistry and instrumentation. First, the ontology provides semantic enhancement to each database object through meta-data, standard vocabularies and data object relationships. There is a one-to-one correspondence between the database objects and the ontology objects. Another use of the ontology is to provide a data-driven model for the creation, provenance and versioning of database objects. One aspect of this is the use of domain specific templates to guide the construction of the database objects. The definition of each database object is in a hierarchy of catalog objects, record objects and components (using the DCAT ontology model). Within each of these object definitions is a link describing how a create a set of automatically generated RDF objects within the CHEMCONNECT database. The RDFs facilitate searching the database. To facilitate versioning, data source tracking and data quality control, operations on the database are organized as transactions. In CHEMCONNECT a transaction has a one to one correspondence with the underlying JAVA operation in the implementation. Within the transaction definition, the set of prerequisites and the output of the operation is defined. The use of transactions helps organize and give semantic enhancement to the set of individual operations within the implementation. The work in this paper is on-going and as the first use-case is concentrating experimental and theoretical information in the chemical domain. The implementation is written in JAVA and is using Google Cloud firestore as the database.


1 INTRODUCTION

CHEMCONNECT is database application within the chemical and instrumentation domain. The motivation for using ontologies within CHEMCONNECT (E. S. Blurock 2019; E. Blurock 2021) stems from the W3C recommendations, ‘Data on the Web Best Practices (Caroline Burle Lóscio 2017). The uses of ontologies within CHEMCONNECT have multiple roles and goals:

- **Ontology Based Data Management:** There is a one-to-one correspondence between ontology objects and (JAVA) data objects. The ontology defines objects (Maali and Erickson 2014), processes (Timothy Lebo, Satya Sahoo, Deborah McGuinness 2013) and transactions (Cicarese et al. 2013). Each object has semantic enhancements

to facilitate provenance, data quality, the use of standard vocabularies, formatting and versioning within the database.

- **Ontology Templates:** The ontology provides templates containing domain specific information that can be inserted into standard database objects within the database. In this way the domain specific knowledge can be enhanced without having to update the JAVA database implementation.
- **Ontology Driven Data Manipulation:** The JAVA implementation interprets and is driven by the ontology. All processes, including transactions, are defined within the ontology. There is a one-to-one correspondence between all the functions provided by the web API and the ontology. The ontology defines the prerequisites

^a <https://orcid.org/0000-0001-9487-3141>

(input) needed and the output expected from the process. The hierarchy of processes and the human and machine-readable vocabularies within the process definitions provide meta-data semantic enhancement. Transactions are a special case of processes used to promote versioning, data tracking and data quality within the database.

- **Static versus Dynamic Knowledge:** The role of the ontology within CHEMCONNECT is to capture ‘static’ knowledge, particularly data structures. The database captures the expanding data and knowledge within the domain.

The ontology gives a semantic context to the data in the database following the principles of *Ontology Based Data Management* (Lenzerini 2011; Dehainsala, Pierra, and Bellatreche 2007). In this respect, the ontology supplements the data within the database by providing additional information about a data object that is fixed for all data objects of the same type. The database object can be seen as an instance of this object.

The human readable meta-data provided by all ontology objects provides documentation, comments and labels that can be used by the user-interface.

Another role of the ontology is as the basis of the data-driven paradigm of CHEMCONNECT. The design philosophy is to minimize the catalog object specificity in the (JAVA) programming by having the definitions within the ontology drive the data object manipulation. Within the implementation, the ontology objects instances are represented as JSON objects within the database and within the JAVA implementation. This and the standard meta-data requirements of each data object promotes domain specific enhancements through ontology development rather than JAVA development. Domain data can be updated in the ontology without any addition JAVA programming.

The information within the ontology can provide:

- **Definition:** The ontology provides the definition of database objects with its sub-parts. The database object is a specific instance of the object defined in the ontology. The basic ontology objects are divided in three types, components, records and catalog objects.
- **Templates:** Templates are generalized information used to fill in domain information, such as data formats, chemical properties, instrument properties, procedure steps, etc., into the general catalog database object instances.
- **Concepts:** This is the hierarchy of domain specific concepts and classifications. The

concepts are also used to fill in domain information in the template.

- **Relationships:** Within the ontology object, RDF relationships are defined to link data within catalog object to facilitate searching for the data. The ontology information is used for the automatic creation of database RDFs. These mapping definitions are provided at every level of the object definition.
- **Transactions:** Transactions are the key to data tracking and versioning. The transaction definition within the ontology defines prerequisite (input) objects, essential defining information, output objects and relationships. The transaction information provides the roadmap to automate the creation, manipulation and ultimately versioning of database objects. Each database object originates from a transaction process. In this way, the entire history of the object, versioning, is documented.

The ontology provides static information common to each data type within the database. The database itself is instances of these abstract objects. The ontology in is used in several capacities.

1.1 Database-Ontology Interaction

The purpose of the ontology definitions is to give semantic context to objects within the database. The ontology represents static information giving definitions, relationships and semantic enhancements to the objects in the database.

```

NameOfPerson
  dcterms:hasPart givenName
  dcterms:hasPart familyName
  dcterms:hasPart UserTitle

```

Figure 1: The ontology definition of NameOfPerson.

For example, the ontological definition of a person’s name (NameOfPerson a record in the ontology with the identifier, foaf:name) says that the name should have three components as shown in Figure 1, where:

- **UserTitle:** A classification representing the title of the person (Mr, Dr. Ms., etc.) with the identifier, foaf:title.
- **givenName:** A string representing the name of the person with the identifier, foaf:givenName.

- `familyName`: A string representing the family name of the person with the identifier, `foaf:familyName`.

A specific database instance of this information, for example the name “Dr. Edward Blurock”, would be represented (as a JSON object) as shown in Figure 2.

```
foaf:name: {
  foaf:title: "Mr.",
  foaf:givenName: "Edward",
  foaf:familyName: "Blurock"
}
```

Figure 2: A specific instance of `NameOfPerson` in JSON format.

Within the database instance, only the ‘essential’ information is given. In the ontology semantic enhancements are given. Within the database instance, the (unique) identifiers point to the corresponding ontology object. For example, `foaf:name` points to the `NameOfPerson` ontology object. The ontology object gives additional (static) information about the name of the person:

- `rdfs:label`: “Name of Person”
- `rdfs:comment`: “The full name of a person (including title)”
- `dcterms:identifier`: “foaf:name”
- `skos:altlabel`: “pname”
- `dc:type`: `NameOfPerson`

In addition, using the `skos:mappingRelation` two database RDF’s (see section `RDFGeneration`) linking the last name and the given name to the catalog object.

2 DATA OBJECT DEFINITION

Within the data object definitions are semantic enhancements promoting the use of descriptive and structural meta-data and data object formatting as recommended by the W3C (Caroline Burle Lóscio 2017).

Identifiers and vocabularies, standard when available, are used within every data object and component. The meta-data within the data object are both machine readable and human readable.

The ontology also provides the machine interpretable formats of each data object, process and transaction. This is also the key to the use of the ontology in a data-driven capacity. The JAVA

implementation interprets the ontology which, in turn, drives the processes.

Through the placement within the ontology hierarchies of classes and subclasses, structural meta-data is provided. Data is within the component, record and catalog structures (Maali and Erickson 2014), templates are within the Concept hierarchy (Miles and Bechhofer 2008), processes are within the `prov:SoftwareAgent` (Timothy Lebo, Satya Sahoo, Deborah McGuinness 2013) and transaction are within the `Event` (Dublin Core 2012) hierarchy.

2.1 Common Catalog Information

The ontology structures have a one-to-one correspondence with data, interface and persistent database structures. There are basically three levels of data structures:

- **Catalog Structures:** These are based on the DCAT Catalog structure (Maali and Erickson 2014). These are the structures representing the main data objects to represent the domain.
- **Record Structures:** Base on the `dcat:record` from the DCAT ontology, these are the records of the catalog. Each record structure contains several pieces of ‘primitive’ information.
- **Components:** These are basically single string primitives that make up the record. Numerical values are strings in the database, but can be interpreted as numerical objects.

Catalog objects are the top-level objects within the database. Both catalog objects and records are compound objects consisting of records and components.

The total catalog object definition is a hierarchy of records and components. In the JAVA implementation and the database, a catalog object is manipulated and stored as a JSON object.

All catalog instances are subclasses of `SimpleCatalogObject` which has the following information:

- `CatalogObjectAccessModify`: This is a reference to which users can modify the catalog object.
- `CatalogObjectAccessRead`: This is a reference to which users can read the catalog object. If this is “Public”, then all users, including guest, can access the information.
- `CatalogObjectKey`: This is a unique key for the catalog object instance.

- **TransactionID:** This is a reference to the transaction (see Section 0) that created the object.
- **CatalogObjectOwner:** This is the owner of the catalog object.

This information is key to determining who has access to the catalog object and how the catalog object was created.

2.1.1 Access Rights

The access rights, meaning who can read, modify or even delete the catalog object, is determined by several keys as shown in Section 0. The keyword in each these fields (including the owner) has the following forms:

- **Username:** This is the username of the account which has the access rights.
- **Consortium:** This is a list of usernames have the same access rights to a set of objects.
- **Public:** This is everybody.

If several user accounts can access the accounts, then a consortium is built. The consortium keyword points to a list of usernames. If the access is a consortium keyword, then the specific user account must be in the list.

In searching through the database, part of the search expression involves joining all the possible combinations of access rights to the `CatalogObjectAccessRead` field. The list of valid consortiums is formed by those which include the user account. Basically, an OR operation with this list of consortiums and the username is appended to the rest of the search expression.

2.2 Semantic Enhancements

The standard basic information associated with every ontological object representing data is:

- **Labels (`rdfs:label`) and Comments (`rdfs:comment`):** These are human interpretable strings that give semantic enhancement to the data. These are also useful in the GUI or in human readable printout.
- **Identifiers (`dcterms:identifier`):** This is a unique ontological identifier specifying that what follows is the specific data type.
- **Alternative label (`skos:altlabel`):** This is a short label uniquely identifying the data type.
- **Type (`dc:type`):** This is the pointer to the datatype of the object.

2.3 Templates

The ontology in CHEMCONNECT provides templates to help build and fill in the information in the catalog objects. There are several important classes of templates:

- **Choices and Classifications:** For a given parameter there could be a list or tree hierarchy of possible choices.
- **Domain Information:** The database catalog object is designed to be very general. The domain specific information within the ontology is used to fill in and structure these general catalog data objects.
- **Transactions:** This is a set of templates for operations on the database (see Section 0).

An example of domain specific information is the specification needed for a scientific instrument. In CHEMCONNECT, a device is viewed as a system of subsystems. Within a system definition there are the set of sub-systems (each with its own system definition), concepts and keywords associated with the system's purpose and domain, and finally a set of parameters describing specific attributes of the device. The set of these attributes are designed by the domain experts as being important characteristics to distinguish, for example, the 'same' instrument from one lab from another. For a particular system these attributes could be, for example, dimensions, configuration, operating ranges, etc.. These attributes are a condensation and machine-readable form of the information found in the 'Experimental Setup' section of a scientific paper.

3 RDF GENERATION

One type of database object is a Resource Description Framework (RDF) triplet. The database RDFs are not static like the ones in the ontology definition, but grow with the addition of database objects. A corresponding RDF database instance (a subclass of `RDFTriple`) is added to the database using the information within the data object. The RDF definition defines how to create a link between two pieces of data within the catalog object. Part of the transaction process is, after the catalog object is created, to create the corresponding RDFs.

The purpose of the database RDFs is to facilitate searching and to link up database object instances.

We view the RDF to be an object linked to a subject by a predicate:

Object -> Predicate -> Subject

```
DatabasePerson
  dcat:record PersonalDescription
  dcat:record FirestoreCatalogID
  dcat:record CatalogObjectKey
  .
  .
  .
```

Figure 3: Excerpt from DatabasePerson ontology object.

```
PersonalDescription
  Dcat:record NameOfPerson
  .
  .
  .
```

Figure 4: Excerpt from PersonalDescription Ontology.

Within the ontology RDF definition, a subclass of `RDFMappingDefinition`, the class name is the *Predicate* name. Within this definition the *Object* is identified with `skos:member` and the *Subject* is identified with `prov:entity`. The object pointed to by the *Subject* and *Object* is searched for within the current catalog object and its value is substituted in the `RDFTriplet` object.

Within a source object, which can be a catalog object, record object or even a component object, the RDF defining class is identified with `skos:mappingRelation`. The ontology object that the *Object* and *Subject* refer to are either directly in the source object definition or in the catalog object where the source object is found.

For example, in Figure 3 we see that one of the records (`dcat:record`) in the ontology catalog object `DatabasePerson` is `PersonalDescription`. Furthermore, looking at the definition of `PersonalDescription` in Figure 4, we see that `NameOfPerson` is a record. In addition to the components (`dcterms:hasPart`) of `NameOfPerson` (as seen in Figure 1), there is an additional mapping with the identifier `skos:mappingRelation` to the RDF object, `RDFPersonFamilyName`, with the elements shown in Figure 5. Since there are two `skos:member` links, this produces two `RDFTriplet` objects. The first has the following form:

- Object: `CatalogObjectKey` (a component)
- Predicate: `RDFPersonFamilyName`
- Subject: `familyName` (a component)

Since both the *Object* and *Subject* are simple component strings, the catalog object that is produced is `RDFSubjectObjectPrimitive` (see Figure 6) where the *Object*, *Predicate* and *Subject* are stored in

the `ShortStringKey` (with identifier `foaf:LabelProperty`), `RDFPredicate` (with identifier `RDFpredicate`) and `RDFSubjectClassName` (with identifier `rdsubjectclassname`) fields, respectively.

```
RDFPersonFamilyName
  skos:member CatalogObjectKey
  skos:member FirestoreCatalogID
  prov:entity familyName
```

Figure 5: Excerpt from RDFPersonFamilyName.

```
RDFSubjectObjectPrimitive
  dcterms:hasPart ShortStringKey
  dcterms:hasPart RDFPredicate
  dcterms:hasPart
```

Figure 6: Excerpt from `RDFSubjectObjectPrimitive` ontology.

Using the example that the name of the person is “Dr. Edward Blurock”, the `RDFSubjectObjectPrimitive`, a catalog object, that would be formed (in JSON form) is shown in Figure 7.

```
RDFSubjectObjectPrimitive {
  foaf:LabelProperty: "Blurock"
  RDFPredicate: "RDFPersonFamilyName"
  rdsubjectclassname: "catalogkey"
}
```

Figure 7: Excerpt from `RDFSubjectObjectPrimitive` database object. The entry “catalogkey” represents the actual unique key for the database object.

The other RDF that is formed is different in two ways. First it involves a record object as the subject, namely `FirestoreCatalogID`. The second is that this record is not a member of `NameOfPerson`, but is found in another place in the total catalog object `DatabasePerson` (see Figure 3). But since the identifiers are unique, the database object needs only to be systematically searched for the corresponding element.

Since the subject is a record, it would produce a `RDFSubjectPrimitiveObjectRecord` object, schematically shown in Figure 8 where the record object is identified as `dcat:CatalogRecord` and the actual record for `FirestoreCatalogID` is identified with `firestorecatalog`.

```
RDFSSubjectObjectPrimitive {
  foaf:LabelProperty: "Blurock"
  RDFSPredicate: "RDFPersonFamilyName"
  dcat:CatalogRecord: {
    firestorecatalog: {
      .
      .
      .
    }
  }
}
```

Figure 8: Schematic of JSON object RDFSSubjectPrimitiveObjectRecord.

3.1 Searching RDFs

The objects in the database are in a complex hierarchical structure. Unless one knows where in this structure the desired object is within the hierarchy, finding the object may be difficult. The purpose of the database RDFs is to provide an efficient search mechanism to find, primarily through keywords, objects in the database.

The RDF definitions in the ontology as outlined in the previous section provides an automatic mechanism to facilitate simple searches through key objects in the database. The RDFs which are generated should reflect useful and efficient searches of the database. For each data object, especially catalog objects, the designer should decide how the object will be accessed and what is the most efficient way to access this information. For example, which keywords within the information within the catalog object can be used to access the information.

In the previous example, an RDF was made linking the last name, "Blurock", with the full information of the user (DatabasePerson). In this case the keyword "Blurock" would be searched in all the Object fields of the stored RDF, the foaf:LabelProperty of the RDFs. The question this RDF answers is 'Find me all the users with the last name of *Blurock*'.

4 TRANSACTIONS

Operations on the database, such as creation, modification or deletion, are defined within CHEMCONNECT as *transactions*. The main motivation of the use of transactions (a sub-class of Event, in the dublin-core ontology (Dublin Core 2012)) is to satisfy the W3C requirements (Caroline Burle Lóscio 2017) for versioning. A CHEMCONNECT transaction has an associated

process (function), the list of prerequisite transactions for the process and the output object of the process. The tree of transactions gives the exact history of the data. There are transactions for the creation and also the manipulation, transformation and updating of data objects. Data quality can be assessed through the transaction history because the sources can be traced.

Within the implementation there is a one-to-one correspondence between an operation on the database and a transaction. The transactions keep track of what is needed to perform the operation and then what catalog objects are created by the operation.

Defining operations as transactions gives each operation an organizational and semantic context. Database modification through transactions also gives the history and dependence of the catalog objects. If an object is dependent on another object which has been modified, there is the possibility to reflect the modification of the modified object on those objects which are dependent on them.

A transaction is defined within the ontology hierarchy in the ontology as a subclass of `dc:Event`. The transaction definition within the ontology has the following elements:

- The set of prerequisite transactions that need to be performed before the current transaction can be executed (`dcterms:requires`).
- An additional data object having the input information needed to perform the current transaction (`dcterms:source`).
- The catalog object that the transaction produces (`hasOutput`).

The prerequisite transaction information can be used to find the output data (`hasOutput`) from previous operations needed to process the current operation.

The additional data (`dcterms:source`) links to a data object giving additional information, not found in the prerequisites, needed to perform the operation. This information can, for example come from the user interface.

```
CatalogObjectUserAccountEvent
  dcterms:requires
    CreateDatabasePersonEvent
  dcterms:source
    ActivityCatalogUserAccount
  hasOutput: UserAccount
```

Figure 9: Excerpt from CatalogObjectUserAccountEvent.

For example, Figure 9 shows the fields of the CatalogObjectUserAccountEvent that is

needed to create a new user account. This event requires that the user (`DatabasePerson`) already exists and this is ensured by requiring that a specific `CreateDatabasePersonEvent` has already been executed. The corresponding database object has some of the information needed, such as the person's name and other details, but there is some extra information needed. This is provided by the link (`dcterms:source`) to an `ActivityCatalogUserAccount` record (see Figure 11: Excerpt from `TransactionEventObject`). All of these records containing the extra information for transactions are found as a subclass of `ActivityInformationRecord` which, in turn is a subclass of `dcat:CatalogRecord`. In this example, this is information needed for the new account that is not found in the `DatabasePerson`.

```
ActivityCatalogUserAccount
  dcterms:hasPart AuthorizationType
  dcterms:hasPart username
```

Figure 10: Excerpt from `ActivityCatalogUserAccount`.

Within the ontology, the general transaction is defined. In the ontology definition, the ontology object class is pointed to (by the identifiers shown in parenthesis above). After each transaction, the catalog object instance, `TransactionEventObject` (see Figure 11), is created. For a specific transaction instance, specific objects of that class within the database are pointed to (again by the same corresponding identifier). The three records pointed to are links, through database IDs, to the required transaction instances, the addition information instance and the output instance. These IDs are enough to find the respective information. From the transaction instances, when needed, the respective output objects from these transactions can be retrieved.

```
TransactionEventObject
  dcat:record
    RequiredTransactionIDAndType
  dcat:record
    ActivityInformationRecord
  dcat:record
    DatabaseObjectIDOutputTransaction
```

Figure 11: Excerpt from `TransactionEventObject`.

4.1 TransactionID

Every catalog object created has the transaction ID as one of its fields. This provides information about the objects origins and history through the creating transaction and its dependencies through the chain of prerequisites found in the transactions. It also can provide a search tool for finding all the objects created by the transaction.

5 CONCLUSION

This paper has outlined several aspects of the ontology-based database CHEMCONNECT. The ontology provides information about the database object instances. The ontology provides semantic enhancement of database objects through annotations and relationships defined within the ontology. The ontology also provides, as in the case of RDF triplet generation, an automation of the database object creation. Through transaction definitions, the ontology also provides a semantic context and organization to the operations of database management.

This work is on-going and in a preliminary phase. The use-case domain is chemical kinetics and physical organic chemistry.

REFERENCES

- Blurock, Edward. 2021. "Use of Ontologies in Chemical Kinetic Database CHEMCONNECT." In , 240–47. <https://www.scitepress.org/PublicationsDetail.aspx?ID=ai7xFiBEN8E=&t=1>.
- Blurock, Edward S. 2019. "CHEMCONNECT: An Ontology-Based Repository of Experimental Devices and Observations." In . Copenhagen, Denmark. <https://icaita2019.org/index.html#home>.
- Caroline Burle Lóscio, Bernadette Farias, Newton Calegari. 2017. "Data on the Web Best Practices." January 2017. <https://www.w3.org/TR/2017/REC-dwbp-20170131/>.
- Ciccarese, Paolo, Stian Soiland-Reyes, Khalid Belhajjame, Alasdair JG Gray, Carole Goble, and Tim Clark. 2013. "PAV Ontology: Provenance, Authoring and Versioning." *Journal of Biomedical Semantics* 4 (1): 37. <https://doi.org/10.1186/2041-1480-4-37>.
- Dehainsala, Hondjack, Guy Pierra, and Ladjel Bellatreche. 2007. "OntoDB: An Ontology-Based Database for Data Intensive Applications." In *Advances in Databases: Concepts, Systems and Applications*, edited by Ramamohanarao Kotagiri, P. Radha Krishna, Mukesh Mohania, and Ekawit Nantajeewarawat, 497–508. Lecture Notes in Computer Science. Berlin,

- Heidelberg: Springer. https://doi.org/10.1007/978-3-540-71703-4_43.
- Dublin Core. 2012. "Dublin Core Metadata Initiative." Dublin Core Metadata Initiative. June 12, 2012. <http://dublincore.org/>.
- Lenzerini, Maurizio. 2011. "Ontology-Based Data Management." In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 5–6. CIKM '11. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/2063576.2063582>.
- Maali, Fadi, and John Erickson. 2014. "Data Catalog Vocabulary (DCAT)." January 16, 2014. <https://www.w3.org/TR/vocab-dcat/>.
- Miles, Alisair, and Sean Bechhofer. 2008. "SKOS Simple Knowledge Organization System Namespace Document 30 July 2008 'Last Call' Edition." August 20, 2008. <https://www.w3.org/TR/2008/WD-skos-reference-20080829/skos.html>.
- Timothy Lebo, Satya Sahoo, Deborah McGuinness. 2013. "PROV-O: The PROV Ontology." 2013. <https://www.w3.org/TR/prov-o/>.

