Technical Report

ISO/TR 22126-2

First edition
2025-01

Financial services — Semantic technology —

Part 2: OWL representation of the ISO 20022 metamodel and e-repository

Services financiers — Technologie sémantique —

Partie 2: Représentation OWL du métamodèle et du référentiel de l'ISO 20022

Reference number
ISO/TR 22126-2:2025(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword v

Introduction vi

1 Scope 1

2 Normative references 1

3 Terms, definitions and abbreviated terms 1
3.1 Terms and definitions 1
3.2 Abbreviated terms 2

4 Overview 2
4.1 Graph transformations 2
4.1.1 General 2
4.1.2 Graphs derived from the XML document 4
4.1.3 Conversion to a FIX AE message 4
4.1.4 Graphs aligned with message and business components 4

5 Methods 5
5.1 Multiple local RDF graphs 5
5.2 Open linked data 6

6 Reconciling differences between Ecore, RDF and OWL 7
6.1 Introducing Ecore 7
6.2 Differences between Ecore and RDF standards 7
6.3 Separation of schema objects and instances 8
6.4 Python, pyecore and RDF 8
6.5 Ordering and structured data in RDF 9
6.5.1 General 9
6.5.2 Unordered relationships 9
6.5.3 RDF collections 9
6.5.4 RDF containers 10
6.5.5 Blank nodes and triples turned sideways 10
6.6 Internationalization in RDF 11
6.7 SHACL, another RDF schema language 11

7 Processing ISO 20022 instance messages in RDF 12
7.1 The G_2 graph — Converting an ISO 20022 XML message to RDF instance data 12
7.2 Querying ISO 20022 message data with SPARQL 14
7.3 The G_1 graph — Maintaining ordering information 14
7.4 Compatibility with OWL and RDFS inference — G_4 graph 14
7.5 The G3 graph — Converting the auth.016 message to a FIX AE message 15

8 Mechanical conversion of the e-Repository 18
8.1 File and namespace prefixes — M_C and B_C graphs 18
8.2 OrganisationIdentification — Business components and attributes 18
8.3 Message components and attributes 21
8.4 CodeSets 23
8.4.1 General 23
8.4.2 Empty CodeSet 23
8.4.3 CodeSet with codes 24
8.5 Coining unique identifiers 25
8.5.1 The problem 25
8.5.2 Composite names 25
8.5.3 Arbitrary disambiguators 25
8.5.4 External id 25
8.5.5 Internal id 26
8.5.6 Random id 26
8.5.7 Skolemization 26

9 Representing the ISO 20022 e-Repository in OWL and RDFS 26
9.1 Methodology 26
9.2 RDFS modelling of a business component 28
9.3 Datatype and object properties in OWL 29
9.4 Representing enumerated types 30
9.5 Restrictions on data types 31
9.6 Disjoint classes 31
9.7 Miscellaneous objects and annotation properties 32
9.8 Containment relationships and inference 32
9.9 OWL inference and its limits 33
10 Instances aligned to the e-Repository 34
10.1 The G_4 graph — Message expressed with message components 34
10.2 The G_5 graph — Abstract instance of a trade 35
10.3 Concepts missing in the business components 37
11 Conclusion 38
Annex A (informative) **A preliminary OWL 2 ontology as a semantic model for ISO 20022** 40
Bibliography 45

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 68, *Financial services*, Subcommittee SC 9, *Information exchange for financial services*.

A list of all parts in the ISO 22126 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The purpose of this document is to demonstrate the feasibility and explore the design space for the representation of the ISO 20022 e-Repository in OWL DL, a decidable dialect of the OWL ontology language. This document demonstrates multiple representations of the auth.016 sample messages illustrating possible modelling choices.

The current e-Repository is constructed in a purpose-built system comprising multiple layers of object-oriented modelling built on the Eclipse Modeling Framework, a specialization of the Essential Meta-Object Facility (EMOF) standardized by the Object Management Group (OMG). This system can create class definitions and object instances in a computer language such as Java[1] that serve as containers for data, but does not define the semantics, or meaning, of those objects. The EMOF model is a foundation to write software on (a "blank slate") but requires custom software to establish the meaning and behaviour of those objects.

In OWL DL, on the other hand, it is possible to construct concepts based on logical definitions. Although custom software can still be necessary, an OWL DL ontology has a meaning and defined behaviours when used with OWL standard tools. Out-of-the-box an OWL ontology works with ontology editors, reasoners and graph databases that allow queries with the SPARQL language. An OWL ontology can be used together with other OWL ontologies. Unlike more expressive systems such as ISO Common Logic, however, OWL DL is based on a subset of first-order logic that is decidable so that in addition to reasoning instances (for instance querying instances that match a logical definition) it can prove the consistency of an ontology. (As OWL is defined in terms of first-order logic, OWL axioms can be exported to other logic-based tools that can be more expressive, such as theorem provers, or more application-oriented, such as business rules engines.)

It is common for schemas and documentation for message families such as ISO 20022, FIX Protocol and SWIFT MT to be written in formats idiosyncratic to the family. Although ISO 20022 messages conform to an XML schema, authoritative definitions exist in the e-Repository as a set of Ecore-serialized objects specific to ISO 20022. The elements of FIX messages (which can be encoded in various wire formats) are defined in FIX Orchestra, an XML file which is specific to the FIX trading community. Tools designed to work with the schema of one standard do not natively work for other standards.

Applications that process financial messages have their own schemas that exist either explicitly or implicitly, for instance a set of tables, constraints, stored procedures and other projects in a relational database, or a hierarchy of classes in a programming language such as Java. Although disparate descriptions for the structure of messages and software systems fulfil the requirements of each system, technologists lack the global view necessary to take rapid and correct action.

To have that global view, it is desirable to bring schemas and instances from disparate systems into a single system which can be queried to support software development, make visualizations and do inference. RDF is a suitable framework for this: not only does it come with standard vocabularies such as RDFS, OWL and SHACL for representing schemas, but it can represent arbitrary data structures as a graph of nodes that possess datatype properties and are interconnected with object attributes. Some major features of RDF are:

a) the XML schema datatypes are available to represent common primitive data types that occur in general computing;

b) nodes and predicates, relationships between nodes, are named with URIs to establish a global namespace in which terms from arbitrary vocabularies can be combined;

c) text strings are tagged with ISO 639 language codes to provide a straightforward mechanism to represent labels and descriptions in multiple languages.

It is possible, as seen in Clause 7, to represent any kind of message in RDF through a simple form of mechanical translation – it possible in RDF to work without a schema of any kind. Clauses 9 and 10 demonstrate that, with more effort, it is possible to express the e-Repository and ISO 20022 messages as an OWL DL ontology.

[1] Java is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

The EMOF form of the e-Repository is built in a number of layers starting from the M3 layer (the foundation of EMOF), the M2 layer (an object model used to describe the M1 model) and the M1 layer (definitions of business components, message components, etc.). Users of the e-Repository can create instances of objects defined at the M1 layer in the M0 layer that define individual messages or business objects. For the most part, the ontology is contained in the M1 layer and defined in terms of an idiosyncratic vocabulary defined in the M2 layer. This vocabulary is fit for purpose, but incompatible with other schema and ontology definition systems.

OWL DL requires a flatter organization where there is a clear separation between ontology objects (definitions of classes and properties) and instance objects (instances of the defined classes.) Most terminology from the M2 layer is replaced with terminology from the OWL standard, making the ontology automatically interoperable with other ontologies. Properties that cannot be expressed with built-in OWL properties, such as XML element names for the message components, are represented with annotation properties which, as defined in the ontology, not participate in inference. Objects in the e-Repository which are not naturally treated as class or property definitions are defined as instances in OWL DL with corresponding definitions derived from the M3 layer.

Users of this ontology can add additional instance and ontology objects. OWL, in particular, allows users to write logical definitions in terms of specific lists of objects, the attributes of objects and logical combinations such as AND, OR and NOT thereof. This makes it possible for a business or regulator to define a particular kind of transaction (a subclass) in terms of size, the time it took place, the security traded, the attributes of the organizations involved, etc. Annex A demonstrates that many categories defined and discussed in regulation and documents are straightforward to express in this manner, a significant step towards making business documents interpretable by machines.

Financial services — Semantic technology —

Part 2:
OWL representation of the ISO 20022 metamodel and e-repository

1 Scope

This document is concerned with the representation of the ISO 20022 e-Repository contents in RDF and OWL by developing a case study around the ISO 20022 auth.016 sample message (hereafter simply referred to as “auth.016”). This includes:

a) transformation of the sample message into an RDF instance graph;

b) demonstrating a set of SPARQL rules that transform the auth.016 message into a FIX TradeCaptureReport(35=AE) message (hereafter simply referred to as “FIX AE”);

c) expressing the metamodel, business components and message components exactly with a custom RDF vocabulary;

d) representing those schemas as OWL schemas using OWL vocabulary when possible and annotation properties otherwise;

e) creating instance graphs for the auth.016 sample messaging using the vocabulary of the business components and message components.

This document also discusses the choices that arise in structuring RDF documents equivalent to documents in XML, and FIX Tag-Value format balancing considerations such as preserving the order of parts of the message versus creating graphs that are suitable for RDFS and OWL inference.

2 Normative references

There are no normative references in this document.

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.2 Abbreviated terms

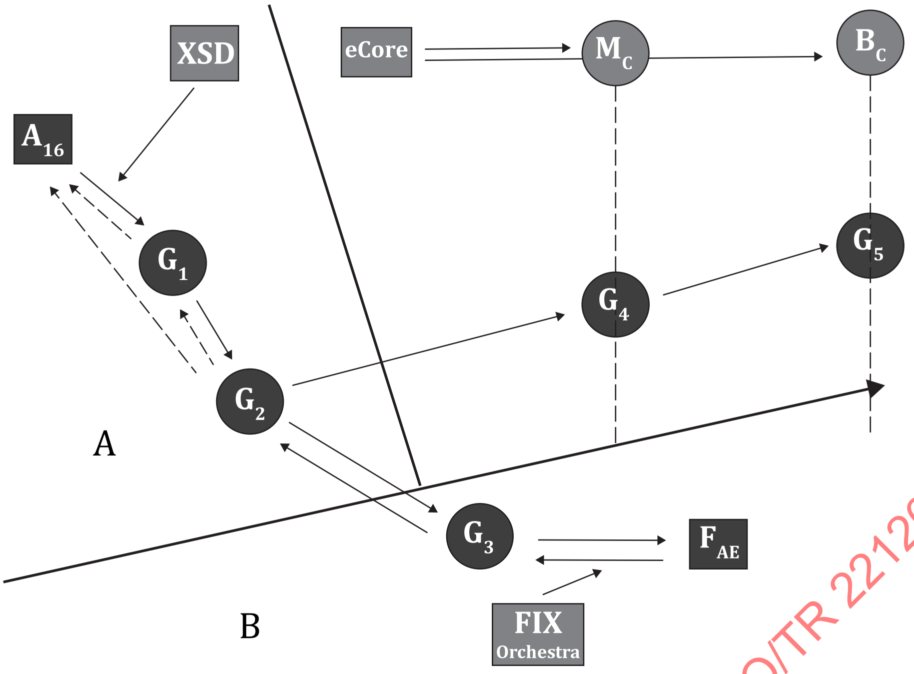
API	application programming interface
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
EMOF	Essential Meta-Object Facility
FIX	Financial Information Exchange
JVM	Java Virtual Machine
LEI	Legal Entity Identifier
MOF	Meta-Object Facility
OMG	Object Management Group
OWL DL	Web Ontology Language – Description Logic
RDF	Resource Description Framework
RDFS	RDF Schema
RMI	Remote Method Invocation
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SWIFT MT	SWIFT Message Type
SWRL	Semantic Web Rule Language
UML	Unified Modelling Language
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XML	Extensible Markup Language
XSD	XML Schema Definition

4 Overview

4.1 Graph transformations

4.1.1 General

Figure 1 depicts the data transformations described in this document.



Key

A auth.016 XML message

B FIX AE message

C graphs aligned to message and business components

A_{16} sample auth.016 message as XML document

G_1 A_{16} translated mechanically into RDF retaining element order

G_2 simplified graph that removes inessential information such as element ordering from G_1

G_3 a FIX AE message equivalent to A_{16} expressed as key-value pairs in RDF

F_{AE} G_3 converted to a complete FIX AE tag-value message

G_4 G_2 rewritten using vocabulary defined in M_c (message components)

G_5 G_4 rewritten using vocabulary defined in B_c (business components)

M_c message Components Ontology in OWL

B_c business Components Ontology in OWL

NOTE Circles represent RDF graphs. Squares represent data in some other format. Blue lines represent transformations that have been implemented. Red lines implement transformations that remain unimplemented. Blue objects represent instance information (a message). Green objects represent schema and ontology information.

Figure 1 — Data transformations of RDF to and from other formats

A_{16} is the auth.016 sample message in XML format. In Clause 7, this message is represented as the G_1 and G_2 graphs which are close to the structure and vocabulary of the XML document. Furthermore, a set of production rules are demonstrated that can convert the A_{16} message into an equivalent F_{AE} FIX AE message, a practical data transformation task.

Clauses 8 and 9 develop representations of the message components M_c and business components B_c in RDF. Clause 8 represents the contents of the e-Repository translated mechanically while Clause 9 represents the contents of the e-Repository in OWL.

In Clause 10 the message is converted to the G_4 and G_5 graphs. The G_4 graph precisely represents the A_{16} content using vocabulary from M_c, a process which can be mechanically implemented. The final destination is the G_5 graph which uses vocabulary from $B_{c,}$ which is shared between multiple messages. Complete capture of the message's meaning, however, requires additional terms that do not exist in B_c.

4.1.2 Graphs derived from the XML document

The method used to create G_1 is mechanical and can be tuned to convert a wide range of XML documents to RDF graphs.

Graph G_1 was produced by combining the A_{16} sample message with information from the XSD schema. XML element and attributes were rewritten to RDF property names by concatenation: for instance, an element named `<Pric>` is rewritten as the property `ns:Pric` where is `ns` is a chosen RDF namespace prefix. Leaf nodes of the XML document are written as datatype properties while upper nodes are connected with object properties. The XSD schema provides default values and resolves question such as “does this instance of the string `‘55’` represent an integer or a character string?”.

G_1 retains information about the order of elements in the document as well as information about where nodes (in terms of line number and character position) appear in the source document. As the order of elements is not essential to the meaning of the message, graph G_2 is a simplified graph that strips away information idiosyncratic to the source document.

4.1.3 Conversion to a FIX AE message

The FIX AE message is equivalent to the auth.016 in meaning and purpose. The G_3 graph represents the meaning of the auth.016 sample message in a vocabulary based on the FIX tag-value standard. A FIX tag such as `35` is written with the predicate `fix:f35` where `fix` is a chosen namespace, `f` means “field” and `35` is the tag number. As with the XML document, the tree structure of the FIX message is represented as a tree of RDF nodes.

The G_3 graph was created from the G_2 graph by matching patterns in G_2 to equivalent patterns in G_3, a process that can be performed in either direction. The process of creating G_3 does most of the work to creating a FIX message equivalent to A_{16}. The remaining task to write a tag-value FIX message is to write tag-value pairs from G_3 the right order, calculate a checksum, append headers and attend to other details based on information from FIX Orchestra.

4.1.4 Graphs aligned with message and business components

Although the G_2 graph captures enough meaning to translate the message, it is unsuitable for RDFS and OWL inference. For example, A_{16} represents the price at which a trade took place such as:

```
<Pric>
     <Pric>
          <MntryVal>
               <Amt Ccy="EUR">13.5</Amt>
         </MntryVal>
     </Pric>
</Pric>
```

Specifically, the element `<Pric>` embeds another `<Pric>` element inside itself. auth.016 allows the outer price element to contain either a `<Pric>` or a `<NoPric>` element depending on whether a price is specified. As a result, in the G_1 and G_2 graphs the `ns:Pric` predicate is used in two different parts of the graph with two different meanings. This is allowed in RDF (SPARQL queries can be written against such a graph) but RDFS and OWL expect that the same meaning is intended wherever a predicate is used.

The G_4 graph is structurally similar to the G_2 graph and continues to represent the complete content of the message. The main difference is that G_4 is written in a vocabulary based on the message components, which, unlike the elements of the XML document, have a context-independent meaning. The same vocabulary is used in the M_c OWL schema compatible with the G_4 graph.

The G_4 graph represents messages accurately at the cost of:

a) not sharing concepts between messages;

b) having little semantic information about the meaning of message parts.

For instance, multiple messages in the e-Repository concern concepts such as financial transactions and assets. In the message components, different objects are used to represent different financial transactions in different messages. As such, every message part can be said to have precisely the meaning it is expected to have in that message.

Some objects from the e-Repository (such as message definitions, message components and choice components) are represented as OWL classes, while other objects (such as message attributes, message association ends and message building blocks) are represented as OWL properties. While translating and XML document, properties are used to represent XML elements and attributes while classes represent the data types contained inside elements and attributes.

The G_5 graph writes the auth.016 message in terms of the business components, which are more general than message components and can be reused between messages. Aligned with the M_c graph, the G_5 graph continues the process of discarding idiosyncrasies of the message in favour of the essential meaning of the message. The cost of this, however, is that the business components are insufficient to capture 100 % of the meaning of the message. (Ideally the G_5 graph can be used for message processing tasks such as creating the G_3 graph to write a FIX message, however, the lossless nature of G_2 made translation G_2 directly to G_3 a safe approach to the translation task.)

NOTE The e-Repository contains data objects (such as such as Business Area and a Message Definition Identifier) which are not naturally represented as properties and classes records and classes. Objects of this type can be represented as RDF instances conforming to classes and properties imported from the metamodel. Objects that map to OWL schema objects have properties such as XML tag names and registration status that do not exist in the OWL vocabulary, but these can be added to the OWL ontology in the form of annotation properties to capture 100 % of the content of the e-Repository.

Annex A considers an alternative approach to representing the business components in which message-related objects from the e-Repository are exclusively and consistently represented as objects as opposed to the alternating classes and properties in the G_4 graph. In this representation, a single property, denoting that one message part directly contains another message part, is the only property used to build message instances. At the cost of introducing additional nodes, this representation can capture the sequencing of message parts in both the schema and instances by adding sequence numbers to the RDF nodes.

5 Methods

5.1 Multiple local RDF graphs

Much of the work described in this document was done using in-memory graphs from the Python[2] rdflib library. Like a client-server triple store (e.g. OpenLink Virtuoso, Ontotext GraphDB, AllegroGraph[3]), an in-memory graph can be queried with SPARQL. The in-memory graph also allows direct interaction with individual triples and nodes which can be helpful when working with list and tree structures. The Jena framework provides similar in-memory graphs which are used internally with the Protégé ontology viewer and editor.

In-memory graphs scale well for graphs in the range of 1 to 10 million triples and are adequate for handling schemas the size of the ISO 20022 e-Repository; a system that processes instance messages can have a few large graphs that for schemas and large number of small graphs which contain individual instance messages.

Unlike the client-server triple store, in-memory graphs are lost when the application shuts down. If the process for creating the graph is repeatable, it can be constructed from the source material as necessary. Alternately, the graph can be serialized to a Turtle[1] or other RDF format file and later restored or exported to other RDF tools. Client-server triple stores can handle much larger data sets than in-memory stores (billions of triples) and can persist triples for long periods of time. Most client-server triple stores work on

[2] Python is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

[3] AllegroGraph is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

RDF data sets, which can contain a number of named RDF graphs, a mechanism to represent a collection of multiple RDF graphs.

Working with structures involving blank nodes can be awkward with a client-server triple store. One issue is that complex structures are traversed in a way that requires many round trips between the client and server with long latencies compared to an in-memory graph. Although most client-server databases provide facilities to work with blank nodes, those facilities are not completely portable and can require adaptation to work with various triple stores.

A client-server triple store applies to a situation where multiple parties are making changes to a graph through an API: for instance, a system that provides a web interface to edit an ontology.

5.2 Open linked data

Open linked data is a major trend in the RDF world, which centres around the use of http-based RDF resources such as `<https://dbpedia.org/resource/Leipzig>` which corresponds to the Wikipedia[4] page for the German city. It is possible to retrieve this document using an http client such as curl. In that case, the DBpedia web server returns a document with facts extracted from Wikipedia about that city in a format such as Turtle or RDF/XML[2] as specified in the `Accept` http header.

Linked data provides a simple way to browse a data set: instead of downloading all facts in a graph (which can contain billions of facts as in the case of DBpedia) it is possible to traverse the graph subject-by-subject and selectively retrieve only the necessary facts out of a large database. A global view can be had efficiently by downloading the entire database and loading it into a triple store, but linked data are practical when it is necessary to traverse just a small part of the graph.

RDF ontology projects can interact with linked data in two ways:

a) a project can incorporate data published in linked data form;

b) it can publish linked data.

This document is based on data entirely from the e-Repository, sample messages and FIX Orchestra and did not require additional linked data.

Linked data are a potential information source for research and development. For instance, concepts from the e-Repository can be aligned to concepts in generic databases such as DBpedia.

Linked data poses risks for applications, such as production software that works with financial message. Linked data, for instance, depends on a working internet connection and linked data services being online. Linked data can be updated without warning, and such changes can break an application. The continuing function of an application depends on the availability of data and requires that the linked data publishers maintain their service indefinitely. Linked data can be curated together with other data to produce operational graphs which are inspected, tested, versioned and proven fit for purpose.

The graphs described in this document are not published on the web and thus primarily use URIs that are non-resolvable, such as

```
<urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
```

The urn scheme is a global namespace with prefixes uniquely registered amongst organizations, as does the http scheme. Unlike the http scheme, however, there is no expectation that a urn URI be resolvable.

If and when an official and stable RDF translation of the e-Repository is ready, that translation can use http URIs and have the facts published on the web. In fact, this option is open to any organization which wishes to publish their own version of the ISO 20022 e-Repository, or that wishes to publish their own instance documents using vocabulary defined in the e-Repository.

[4] Wikipedia is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

6 Reconciling differences between Ecore, RDF and OWL

6.1 Introducing Ecore

The ISO 20022 repository is represented in Ecore, an implementation of Meta-Object Facility (MOF) which corresponds closely to the EMOF[3] dialect of MOF. Ecore was created for the Eclipse Modeling Framework to represent Java class hierarchies as part of their CASE tool. MOF, more generally, is intended to be the foundation to define UML and applications of UML by bootstrapping from a small core.

MOF uses a simple vocabulary (M3) to define a richer vocabulary, such as UML 2 (M2). The M1 model is written in the vocabulary defined in the M2 (this can be a UML model that describes a set of Java classes) and the actual set of Java classes generated by a CASE tool (that which is modelled) comprise M0.

This layering is used in the ISO 20022 e-Repository, which uses the M3 vocabulary from MOF to write a M2 metamodel that defines the vocabulary used to describe business areas, messages, message components, business components and similar objects. The message definitions and related concepts comprise the M1 model. Finally, the M0 level contains the set of ISO 20022 message instances, objects representing those instances and automatically generated software (e.g. Java classes) that support working with that representation.

Ecore is focused on modelling objects from the Java programming language (generics are supported directly in M3); however, MOF in general is consistent with the OMG’s goal of being able to access an object on another server (as in RMI, CORBA or DCOM) or to otherwise provide access to objects in a foreign system (such as access a Java object from a C program in the same address space).

6.2 Differences between Ecore and RDF standards

Ecore looks similar to OWL and RDFS in that it contains similar-sounding concepts such as classes, properties and relationships. Ecore and OWL similarly makes a distinction between object properties (relationships between two instance objects such as the relationship between a trade and the participant of a trade) and datatype properties (an attribute which is expressed in terms of primitive types such as strings and numbers such as the LEI of a trade participant).

Seen as documentation, Ecore models and OWL schemas look similar, yet the meaning of the models and schemas is quite different.

An Ecore schema is a blueprint for defining classes in a statically typed language such as Java. In Java, class definitions are defined and compiled to class files. Instances are only created later when those classes are loaded into a JVM and those classes are constructed. In Java, the definition of fields in the source code allocates a location in memory where attributes are stored so it is not possible to add new attributes at run time.

In RDF, on the other hand, it is not necessary to have a schema at all to process data. Arbitrary triples can be inserted into the graph with arbitrary URIs for subjects and predicates at any time. Arbitrary triples can be queried with SPARQL, traversed with a graph library or serialized to a format such as Turtle. A particular graph can fail to validate with SHACL, OWL or other tools. The ontologies described in Clause 9 and Annex A conform to the requirements of OWL DL, but general RDF graphs can use any vocabulary whatsoever.

In OWL DL, the graph is conceptually divided into two parts: the T-Box and the A-Box. Ontology objects such as classes and properties comprise the T-Box or “terminology component” and the instance data described by the ontology is the A-Box or “assertion component”. An RDFS or OWL schema is a set of T-Box axioms that enable an inference engine to discover new facts based on those axioms interacting with facts in the A-Box. OWL additionally is capable of doing reasoning on the T-Box in isolation, determining facts that must be true about any instances that conform to an ontology, or proving that an ontology is inconsistent.

Here is a simple example of RDFS inference. From this set of RDF facts:

```
:John:hasChild :Bill .
:hasChild rdfs:domain :Parent .
:hasChild rdfs:range :Child .
```

an RDFS engine infers:

```
:John a :Parent .
:Bill a :Child .
```

This is backwards from the way Ecore and most programming languages work. In Java, for instance, the type of an object is determined when it is created. With RDFS and OWL, the type of an object can change when attributes are added and removed.

Apart from the documentation value, RDF and particularly OWL inference must be considered like a programming language to be useful. A given set of axioms, combined with a given set of facts, gives certain results. To get the desired results, it is necessary to design an appropriate set of axioms and organization of the data. Furthermore, different OWL reasoners support different profiles of the standard that offer different trade-offs between scalable performance and features: an OWL ontology can require specialization for both the application and the tools in use.

Furthermore, it is a common situation in OWL for ontology users to define new classes and properties based upon logical definitions. In a role similar to a query, a new class can be defined in terms of a restriction, such that members of a class have particular values for particular attributes. For instance, a trade involving a quantity of fewer than 100 units can be defined as an “odd lot trade”. The value of an ontology is not just in specifying possible relationships but in being a foundation for logical modelling of the domain.

6.3 Separation of schema objects and instances

It is central to EMOF and Ecore that schemas are defined in layers. For instance, there is a schema describing the basic objects of Ecore (M3). Instances of those objects define the e-Repository metamodel (M2) which is itself a schema, instances of which are used to define messages (M1), which themselves can be used as a schema over instances of those messages (M0).

OWL schemas that blend schema and instance objects, however, run the risk of being undecidable. In practical OWL dialects such as OWL DL, schema objects are distinct from the instance objects defined in the schema. It is not legal to apply a property defined in the schema to a schema object. For instance, if this is written:

```
:MyClass a owl:Class .
:MyProperty a owl:ObjectProperty .
```

it is forbidden to write

```
:MyClass :MyProperty :MyValue .
```

but it is allowed to write this triple if instead `:MyProperty` is defined as an annotation property:

```
:MyProperty a owl:AnnotationProperty .
```

An annotation property does not participate in inference but it does allow statements to be asserted about schema objects for documentation purposes and also interpretation by software other than OWL reasoners.

There is a conceptual problem in mapping the multiple layers of the EMOF model to OWL in that an EMOF object can be an instance object seen from one level but a schema object seen from another. Any mapping from an EMOF model to OWL requires a disciplined approach to how objects from various labels are represented or not represented in the OWL.

The main content of the e-Repository consists of instances from the M1 model such as message components and business attributes that are represented as schema objects in OWL. Certain objects from the M1 level, however, can be represented as instances in OWL, and schema definitions from those objects can be imported from the M2 level to OWL. Definitions at the M3 level and some definitions from the M2 level can be ignored entirely, but other objects from the M2 level can play a useful role in the OWL model.

6.4 Python, pyecore and RDF

This subclause describes a custom tool that was developed during the research that led to this document.

Conventional MOF tools (such the pyecore library) usually generate a set of objects in the host programming language that make it easy to manipulate objects. Such tools can define a method such as:

```
anObjectInstance.getSomeProperty()
```

where anObjectInstance is an EMOF object realized in the host language.

The custom tool developed for this project generates a set of functions (as opposed to methods) that work similarly.

Instead of representing the EMOF object as a native Python object, the object is represented as a node in an RDF Graph, namely aReferenceToObject. Instead of defining a method on an object, the tool creates a function named `getSomeProperty` that can be called on that reference to get the value:

```
getSomeProperty(aReferenceToObject)
```

This function looks up the properties of the object in the RDF graph using methods from rdflib and computes the appropriate result based on the graph. This is an example of accessing a foreign object (an RDF graph) through code-generated stubs and makes possible a convenient style for accessing the RDFized Ecore model as Ecore was intended to work.

6.5 Ordering and structured data in RDF

6.5.1 General

To completely capture the information in messages and schemas it is necessary to represent the order of items in RDF.

6.5.2 Unordered relationships

Much of the time, the ordering of a collection of objects in a computer does not matter. For instance, in a conceptual ontology, there is no significance in the ordering of attributes of a concept.

The common way to write facts in RDF is simple triples, such as:

```
:Payment :hasAttribute :id, :sender, :recipient, :amount .
```

The attributes of :payment can be queried in SPARQL[4] such as:

```
select {
    ?field
} where {
    :Payment:has Attribute ?field .
}
```

The order that the fields are returned by this query is arbitrary. If the order of objects is significant, however, there are two standards for ordered collections in RDF, i.e. RDF collections (discussed in 6.5.3) and RDF containers (discussed in 6.5.4). Subclause 6.5.5 discusses an additional strategy for ordering information based on general reification of facts with blank nodes.

6.5.3 RDF collections

RDF collections are represented as a Lisp-style linked list. These can be easily written in Turtle as:

```
:Payment :hasField (:id, :sender, :recipient, :amount) .
```

RDF collections are immutable, and so are used in the OWL specification (`owl:AllDifferent` and `owl:distinctMembers`) to specify sets that cannot be extended.

The custom software developed for this project has a Python function `py_list` that converts an RDF collection to a Python list of RDF nodes in order based on the rdflib API.

Collections are a bit awkward to query in SPARQL:

```
SELECT ?field
WHERE {
 :Payment :hasField/rdf:rest*/rdf:first ?field .
}
```

retrieves the fields without respect to the order. With standard SPARQL 1.1, it is not possible to write a query that gets the fields in the right order and it is necessary to traverse nodes one by one to get them in order.

6.5.4 RDF containers

RDF standardizes another ordered data model. The RDF container, which can be written as:

```
:Payment :hasField [
   a rdf:Seq;
   rdf:_1 :id;
   rdf:_2 :sender;
   rdf:_3 :recipient;
   rdf:_4 :amount
]
```

This model is an array instead of a linked list. Like RDF collections, most RDF libraries have functions for converting between native lists and RDF collections. With RDFS inference enabled this list of elements is retrieved (out of order) such as:

```
SELECT ?field
WHERE {
 :Payment:hasField/rdf:member ?field .
}
```

NOTE RDFS treats `rdf:_N` for any integer N as a sub-property of `rdf:member`. Since the index is embedded in the `rdf:_N` it is possible (but awkward) to sort on the field order in SPARQL or to pass the indexes back in the query result to determine the order in client code.

6.5.5 Blank nodes and triples turned sideways

It is possible to add additional data to a property by adding an additional blank node and additional relationships such as `rdf:value`. For instance, the order of properties can be written as a property:

```
:Payment :hasField
   [ rdf:value :id ; :order 1 ],
   [ rdf:value :sender; :order 2 ],
   [ rdf:value :recipient; :order 3 ],
   [ rdf:value :amount; :order 4 ] .
```

It is entirely straightforward to recover the order with a SPARQL query:

```
SELECT ?field
WHERE {
 :Payment:hasField [ rdf:value ?field; :order ?order ] .
}
ORDER BY ?order
```

This is a powerful strategy because it can be used to attach other attributes to a property, such as:

- the time an observation was taken;
- physical units or currency;
- the person who added this fact to the database.

Furthermore, this strategy can be used selectively with particular predicates to solve a specific problem or it can be applied to all of the triples in a particular graph. Software expecting a particular vocabulary (e.g. an OWL reasoner or a SKOS navigation tool) will not work when `rdf:value` statements are interspersed in the graph. Once data has been filtered and processed in the expanded graph it is a simple transformation to remove the `rdf:value` statements and export a graph in the original vocabulary, compatible with conventional tools.

Blank nodes are used extensively in this document for two other purposes:

- creating records comparable to a row in a relational database;
- representing hierarchical message structures.

The content of a payment message can be written as follows in Turtle:

```
[
   a :Payment ;
  :id "0012743" ;
  :sender :Alice ;
  :recipient :Bob ;
  :amount [ rdf:value "75.20"^^xsd:decimal; :currency "USD" ]
]
```

Since the blank node at the root of that message is unnamed, it is accessed by searching for its attributes, such as the `:id` in the following SPARQL query:

```
SELECT ?payment
WHERE {
    ?payment a :Payment; :id "0012743" .
}
```

which returns a blank node object that can be used to access the message record.

The above message uses a second blank node to gather the properties of :amount together, so both the numeric value and the currency can be specified as a single entity. An extended example of this method can be seen in Clause 7.

Flat or tree-structured, a record can be extracted from the surrounding graph by starting at the root and then traversing along triples from left to right, continuing the traversal across blank nodes and stopping the traversal, but retaining the node, on reaching a URI node.

6.6 Internationalization in RDF

One benefit of RDF compared with many data representations is that it supports multiple languages natively. Although the long-term aim of semantic technology is to develop meaningful machine-readable descriptions independent of languages, for the time being it is possible to add multiple instances of labels, descriptions and other text, and tag those instances with ISO 639 language tags. For instance:

```
dbpedia:Sun rdfs:label "Sun"@en, "Sonne"@de, "太陽"@ja, "太阳"@zh .
```

Unlike the situation in MOF, language-tagged strings are standard in RDF and are available in any RDF vocabulary such as the custom vocabulary used in this document, OWL, RDFS, SHACL, SKOS, etc.

6.7 SHACL, another RDF schema language

SHACL is a newer RDF schema language than RDFS and OWL that functions more like a conventional schema (e.g. XSD) in that it enforces that an RDF document is compliant with an expected vocabulary and has an expected structure.[5]

Primarily, OWL is orientated around classes in the sense of classification, i.e. in terms of classes defined in terms of logical definitions. OWL is capable of various feats of validation, but OWL tools frequently do not give good explanations of what causes an error and how to repair it.

SHACL works the way many users expect a constraint language to work and is designed to give clear error messages to quickly resolve common user errors such entering a typo in a URI.

The ISO 20022 e-Repository can be converted to a SHACL schema that validates the instance messages described in 10.1. SHACL can be used in parallel with logical definitions written in OWL or other rules languages such as SPIN[6] and SWRL[7]. SHACL contains its own rule facility, SHACL rules, that is similar to SPIN.

7 Processing ISO 20022 instance messages in RDF

7.1 The G_2 graph — Converting an ISO 20022 XML message to RDF instance data

ISO 20022 XML messages can be converted to RDF using information from the post-schema validation infoset derived from an XML document together with the XSD schema. This clause presents results for the auth.016 message published in the ISO 20022 documentation. Here is the auth.016 message sample XML message from ISO 20022 expressed in Turtle format:

```
@prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/> .
@prefix bare: <http://rdf.ontology2.com/bare-xml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] a :Document ;
 :FinInstrmRptgTxRpt [
    a :FinancialInstrumentReportingTransactionReportV01 ;
   :Tx [
      a :ReportingTransactionType1Choice ;
     :New [
        a :SecuritiesTransactionReport4 ;
          :AddtlAttrbts [
            a :SecuritiesTransactionIndicator2 ;
           :SctiesFincgTxInd [
              a :TrueFalseIndicator ;
              rdf:value true
            ]
          ];
          :Buyr [
            a :PartyIdentification79 ;
           :AcctOwnr [
              a :PartyIdentification76 ;
             :Id [
                a :PersonOrOrganisation1Choice ;
               :LEI [
                  a :LEIIdentifier ;
                  rdf:value "11111111111111111111"^^xsd:string
                ]
              ]
            ]
          ] ;
         :ExctgPrsn [
             a :ExecutingParty1Choice ;
            :Algo [
               a :Max50Text ;
               rdf:value "A123456"^^xsd:string
             ]
          ] ;
         :ExctgPty [
            a :LEIIdentifier ;
            rdf:value "00000000000000000000"^^xsd:string
          ] ;
         :FinInstrm [
            a :FinancialInstrumentAttributes3Choice ;
           :Id [ a :ISINOct2015Identifier ;
            rdf:value "AB1234567890"^^xsd:string
          ]
        ] ;
        :InvstmtPtyInd [
          a :TrueFalseIndicator ;
          rdf:value true
        ] ;
        :OrdrTrnsmssn [
          a :SecuritiesTransactionTransmission2 ;
         :TrnsmssnInd [
            a :TrueFalseIndicator ;
            rdf:value true
          ]
        ] ;
        :Sellr [
          a :PartyIdentification79 ;
         :AcctOwnr [
            a :PartyIdentification76 ;
           :Id [
              a :PersonOrOrganisation1Choice ;
             :LEI [
                a :LEIIdentifier ;
                rdf:value "22222222222222222222"^^xsd:string
              ]
            ]
          ]
        ] ;
       :SubmitgPty [
          a :LEIIdentifier ;
          rdf:value "00000000000000000000"^^xsd:string
        ] ;
        :Tx [
          a :SecuritiesTransaction1 ;
         :Pric [
            a :SecuritiesTransactionPrice4Choice ;
           :Pric [
              a :SecuritiesTransactionPrice2Choice ;
             :MntryVal [
                a :AmountAndDirection61 ;
               :Amt [
                  a :ActiveCurrencyAnd13DecimalAmount ;
                  bare:Ccy [
                    rdf:value "EUR"
                  ] ;
                  rdf:value 13.5
                ]
              ]
            ]
          ] ;
         :Qty [
            a :FinancialInstrumentQuantity25Choice ;
           :Unit [
              a :DecimalNumber ;
              rdf:value 100.0
            ]
          ];
         :TradDt [
            a :ISODateTime ;
            rdf:value "2018-12-17T09:30:47+00:00"^^xsd:dateTime
          ];
          :TradVn [
            a :MICIdentifier ;
            rdf:value "XDUB"^^xsd:string
          ] ;
         :TradgCpcty [
            a :RegulatoryTradingCapacity1Code ;
            rdf:value "MTCH"^^xsd:string
          ]
        ];
        :TxId [
          a :Max52Text ;
          rdf:value "T123456"^^xsd:string
        ]
      ]
    ]
  ] .
```

This Turtle document is structurally similar to the XML sample message, as the paired square brackets [and] that create a tree of blank nodes work like the paired start and end elements in XML. This representation uses numerous blank nodes, particularly as the `rdf:value` predicate adds a level of indirection to literal values at the leaves. It is possible to write:

```
[ :TxId “T123456”^^xsd:string ]
```

without the `rdf:value`, but introducing `rdf:value` reifies this fact. This creates a unique identity (local to the graph) for the fact, to which additional information can be added such as type information derived from the XML schema.

7.2 Querying ISO 20022 message data with SPARQL

SPARQL queries can be written against translated messages in the form described in 7.1. For a regulator that receives auth.016 documents with the intention to produce reports, a collection of G_2 graphs of auth.016 documents can be stored in a triple store and queried with the standard SPARQL language.

For instance, this query:

```
prefix : <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
prefix bare: http://rdf.ontology2.com/bare-xml#prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
select ?txId ?instrument ?qty ?buyr ?sellr
where {
  [
   :TxId/rdf:value ?txId ;
   :Buyr/:AcctOwnr/:Id/:LEI/rdf:value ?buyr ;
   :Sellr/:AcctOwnr/:Id/:LEI/rdf:value ?sellr ;
   :Tx/:Qty/:Unit/rdf:value ?qty ;
   :FinInstrm/:Id/rdf:value ?instrument
 ]
}
```

produces a table of transactions:

```
======= ============ === ==================== ====================
txId    instrument   qty buyr                 sellr
======= ============ === ==================== ====================
T123456 AB1234567890 100 11111111111111111111 22222222222222222222
======= ============ === ==================== ====================
```

7.3 The G_1 graph — Maintaining ordering information

The G_2 graph represents the XML message accurately except that it fails to preserve the exact order of the elements. This information is not necessary for many semantic projects because the order of elements in a message such as the auth.016 message makes no difference to the meaning of the message.

The G_1 graph uses the methods of 6.5.5 to preserve the order as well as the line number and character position of nodes in the source document and additional numbers from the nested set representation described in Reference [8].

Annex A describes a representation of the e-Repository that is different from the G_1 graph in that it uses a single property to state that one message part is contained in another part instead of the many properties in the used in the G_1 graph but is similar to the G_1 graph in that ordering can be represented in both the ontology and in instances by appending sequence numbers. Even if this sequence information is not significant for the meaning of a message, this sequence information is necessary for a system that generates valid ISO 20022 messages.

7.4 Compatibility with OWL and RDFS inference — G_4 graph

The G_2 graph was created directly from the input XML document and is aligned to the names of XML elements and attributes, not to concepts from the e-Repository. This accurately represents the auth.016 message but has unfortunate consequences, including that the same predicate is used to specify properties in different parts of the graph with different meanings. For instance, the predicate `:Tx` is used twice and so is `:Pric`.

This is valid RDF which can be queried with care, for instance the following SPARQL pattern:

```
?tx a :SecuritiesTransaction1 .
?tx :Pric/:Pric/:MntryVal/:Amt/rdf:value ?amount .
```

This works to connect a specific transaction `?tx` to an numerical `?amount` despite `:Pric` being used in two different ways. There is a problem, however, if the schema is aligned with the data, i.e. the multiple kinds of `:Pric` in the document have different attributes, including text definitions, in the e-Repository.

The e-Repository contains sufficient information to assign a specific predicate to each property in the graph. Clause 8 presents a vocabulary that forms URIs based on a combination of the class and property names. As such, the outer use of `:Pric` is written as `en:SecuritiesTransaction1.Price` and the inner use is written as `en:SecuritiesTransactionPrice4Choice.Price`.

7.5 The G3 graph — Converting the auth.016 message to a FIX AE message

A major advantage of RDF is the ability to use an unlimited number of different representations and vocabularies with the same tools (query language, inference, serialization, etc.) One example is the conversion of the G_3 message representing the auth.016 sample message into the G_4 graph representing an equivalent FIX AE message.

The G_4 graph is built from a vocabulary equivalent to the FIX tag-value pair representation. Predicate URIs were created by combining a namespace, a letter denoting a field or a group and the tag number. For instance, the FIX tag 1003 (TradeID) is written as `fix:f1003`.

NOTE One quirk of RDF is that something like, for example, `field:1003`, cannot be written because the local part of a string cannot start with a digit.

In situations where a group of fields repeats, such as the `<TrdRegTimestamps>` component block, the FIX tag `fix:f768(NoTrdRegTimestamps)` is repurposed from being a count of the timestamp blocks to a relationship that links the timestamp entries to the rest of the message.

The translated message in Turtle format looks as follows:

```
@prefix fix: <http://rdf.ontology2.com/fix/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
[] fix:f1003 "T123456"^^xsd:string ;
   fix:f1524 "EUR" ;
   fix:f22 4 ;
   fix:f29 3 ;
   fix:f30 "XDUB"^^xsd:string ;
   fix:f31 13.5 ;
   fix:f32 100.0 ;
   fix:f35 "AE" ;
   fix:f423 2 ;
   fix:f48 "AB1234567890"^^xsd:string ;
   fix:f856 0 ;
   fix:f1116 [
      fix:f1117 "00000000000000000000"^^xsd:string ;
      fix:f1118 "N" ;
      fix:f1119 1 ;
      fix:f1120 [
         fix:f1121 "F" ;
         fix:f1122 64
      ]
   ],
   [
      fix:f1117 "00000000000000000000"^^xsd:string ;
      fix:f1118 "N" ;
      fix:f1119 67 ;
      fix:f1120 [
         fix:f1121 "N" ;
         fix:f1122 81
      ],
      [
         fix:f1121 "Y" ;
         fix:f1122 49
      ]
   ];
   fix:f552 [
      fix:f54 2 ;
      fix:f453 [
         fix:f447 "N" ;
         fix:f448 "22222222222222222222"^^xsd:string ;
         fix:f452 27
      ]
   ],
   [
      fix:f54 1 ;
      fix:f453 [
         fix:f447 "N" ;
         fix:f448 "11111111111111111111"^^xsd:string ;
         fix:f452 27
      ]
   ] ;
   fix:f768 [
      fix:f769 "2018-12-17T09:30:47+00:00"^^xsd:dateTime ;
      fix:f770 1
   ]
.
```

Like the auth.016 message, the Turtle message of the FIX sample message bears a close resemblance to the message written in tag-value format.

RDFS is capable of mapping one vocabulary to another but it is not quite sufficient to convert the G_2 graph to the G_3 graph. The gist of the mapping between vocabularies can be expressed in statements such as:

```
:TxId rdfs:subPropertyOf fix:f1003 .
```

which in RDFS means that the `fix:f1003` property implies the `:TxId` property. The two predicates mean essentially the same thing but they appear in structurally different message graphs.

As opposed to inferring more facts in the same graph, the G_2 to G_3 conversion process builds a new graph by matching patterns in the G_2 graph to equivalent patterns in the G_3 graph.

For the auth.016 sample message, this transformation can be accomplished with a set of 17 SPARQL rules executed in a specific order. SPARQL rules copy and transform patterns in the graph and are equivalent to IF-THEN production rules, except that a SPARQL construct query is written in the opposite order with the consequent first and the antecedent last.

If rules are executed in a circular or recursive way, it can be necessary to use a more advanced rules engine, such as a SPIN implementation or a Rete engine, to implement inference. SPIN applies SPARQL rules repeatedly until a fixed point is reached (all consequences have been derived.) Rete engines such as the Jena Rules language[9] create data structures to efficiently search for rules that match patterns as new triples are added to the database.

The creation of the G_3 graph begins by inserting a new blank node with just the message type. Statements in the `myself:` namespace keep track of the correspondence between the root nodes in particular input and output messages.

```
prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
prefix fix: <http://rdf.ontology2.com/fix/>
prefix myself: <http://rdf.ontology2.com/myself#>
construct {
    ?new fix:f35 "AE" .
} where {
    [ myself:right ?new ; myself:left ?old ]
}
```

Some rules contain table lookups that match up arbitrary concepts on the two sides, such as this rule that inserts the trade report type:

```
prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
prefix fix: <http://rdf.ontology2.com/fix/>
prefix myself: <http://rdf.ontology2.com/myself#>
```

```
# Appendix A Item 1

construct {
    ?new fix:f856 ?fixCode .
} where {
    [ myself:right ?new ; myself:left ?old ] .
    values (?auth16 ?fixCode) {
        ( :New 0)
        ( :Cxl 6)
    }
    ?old:Tx [?auth16 []] .
}
```

Some rules, such as the one that copies the transaction id are remarkably simple:

```
prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
prefix fix: <http://rdf.ontology2.com/fix/>
prefix myself: <http://rdf.ontology2.com/myself#>

# Appendix A Item 2 -- Transaction Reference Number

construct {
    ?new fix:f1003 ?txId .
} where {
    [ myself:right ?new ; myself:left ?old ] .
    ?old:Tx/:New/:TxId/rdf:value ?txId
}
```

Others map a very simple pattern on the ISO 20022 side to a more complex pattern on the FIX side:

```
prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/>
prefix fix: <http://rdf.ontology2.com/fix/>
prefix myself: <http://rdf.ontology2.com/myself#>

# Q:  the "67" is for the investment firm case

construct {
    ?new fix:f552 [
        fix:f54 1 ;
        fix:f453 [
            fix:f448 ?lei ;
            fix:f447 "N" ;
            fix:f452 27
        ]
    ]
} where {
    ?new ^myself:right/myself:left ?old .
    ?old:Tx/:New/:Buyr/:AcctOwnr/:Id/:LEI/rdf:value ?lei .
}
```

This set of rules supports a limited set of auth.016 messages. For instance, it supports only messages that specify parties using the legal entity identifier. Additional rules can be added to support identifiers common to auth.016 and FIX such as a `:MIC` property on the ISO 20022 side is handled much like an `:LEI` property is in the last rule, except “G” is used instead of “N” in field 447 on the FIX side.

In general, there is some mismatch between what can be expressed in two different message formats. For instance, if field 447 in FIX contains a “E” the identifier is an ISO country code (e.g. “UA”) which is not allowed as a party identifier in the auth.016 message.

Almost all the rules in this set are bidirectional in the sense that the CONSTRUCT and WHERE clauses in the SPARQL queries can be reversed to either transform the G_2 graph to G_3 or vice versa.

The main challenge of this approach is that transformation rules are written by hand. The ideal situation is that terms in both vocabularies are written in some machine-readable way such that an inference engine can discover correspondences between those terms.

Automating this is a difficult problem because of the high accuracy required. In full-text retrieval, for instance, 70 % accuracy for the first result is pretty good but the sample FIX message contains 27 unique concepts that must be mapped correctly – even with a 97 % accuracy there would be typically one mistake per message.

Two approaches to conversion are:

a) working on one message at a time, traversing the schemas to find connection each time;

b) working on the schemas and a collection of messages to develop a translation procedure (e.g. a rule set like the one presented above) that can be applied to multiple messages.

The first has the advantage that effort is focused on parts of the schema that really get used, but the second is more interpretable and can be quality controlled more like a traditional piece of software.

8 Mechanical conversion of the e-Repository

8.1 File and namespace prefixes — M_C and B_C graphs

Just as message instances can be converted to RDF mechanically, so can the schema information in the message components and ontology information in the business components. This is straightforward, because an Ecore model is a collection of objects that have both datatype properties (literal values) and object properties (links to other objects.) By naming the objects and predicates appropriately, the Ecore graph is transformed into an RDF graph.

URIs for objects are created by concatenating object names from Ecore to a namespace prefix. Predicates URIs are created by concatenating the type of an object, a forward slash and the name of the property to a namespace prefix, a necessary operation because property names in the e-Repository are not unique. Such a predicate appears in the Turtle file as `:BusinessComponent\/element,` where the backwards slash is necessary to escape the forward slash.

The following namespace prefixes are defined in this file and are necessary to read the Turtle snippets in this clause:

```
@base <https://iso20022.plus/semantic/repository/> .
@prefix en: <en/#> .
@prefix x: <?xmi:id=> .
@prefix: <urn:iso:std:iso:20022:2013:ecore#> .
@prefix iso20022: <urn:iso:std:iso:20022:2013:ecore#> .
@prefix i: <https://www.iso20022.org/standardsrepository/type/> .
@prefix xmi: <http://www.omg.org/XMI#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .
```

8.2 OrganisationIdentification — Business components and attributes

The e-Repository contains two layers of representation of the financial messaging domain: business components and message components. The business components are an abstract ontology of financial concepts, whereas the message components are the schema of concrete messages.

Types such as `:BusinessComponent` and predicates such as `:BusinessComponent\/element` exist in the M2 layer of the MOF, i.e. these are defined in the metamodel for the e-Repository. The components and related objects comprise the M1 layer of the MOF and are the main component of the e-Repository. The:`BusinessComponent` plays a role similar to `owl:Class` and the `:BusinessAttribute` plays a role similar to `owl:DatatypeProperty`.

`:BusinessComponent\/element` in turn plays a role similar to `owl:domain`: A property with predicate:`BusinessComponent\/element` links `en:OrganisationIdentification` on the left to elements such as `en:OrganisationIdentification.LEI` on the right.

Here is a sample business component:

```
en:OrganisationIdentification
 rdfs:label "OrganisationIdentification";
 a :BusinessComponent;
 rdfs:seeAlso i:OrganisationIdentification ;
 xmi:id "_E48d9cTGEeChad0JzLk7QA_-400209460";
 rdf:ID "_E48d9cTGEeChad0JzLk7QA_-400209460";
:BusinessComponent\/name  "OrganisationIdentification" ;
:BusinessComponent\/definition  "Unique and unambiguous way to identify an organisation." ;
:BusinessComponent\/registrationStatus  "Registered" ;
:BusinessComponent\/superType en:PartyIdentificationInformation ;
:BusinessComponent\/derivationComponent en:TradingVenueIdentification1Choice ,
en:SecuritiesTransactionTransmission2, en:TradingVenueIdentification2 ;
:BusinessComponent\/associationDomain en:Organisation.OrganisationIdentification,
en:OrganisationName.Organisation, en:CashClearingSystemMember.OrganisationIdentification ;
:BusinessComponent\/derivationElement en:TradingVenueIdentification1Choice.Other ;
:BusinessComponent\/element
   en:OrganisationIdentification.BICFI ,
   en:OrganisationIdentification.AnyBIC ,
   en:OrganisationIdentification.OrganisationName ,
   en:OrganisationIdentification.Organisation ,
en:OrganisationIdentification.ClearingSystemMemberIdentificationType ,
   en:OrganisationIdentification.BICNonFI ,
   en:OrganisationIdentification.EANGLN ,
   en:OrganisationIdentification.CHIPSUniversalIdentifier ,
   en:OrganisationIdentification.DUNS ,
   en:OrganisationIdentification.BankPartyIdentification ,
   en:OrganisationIdentification.MIC ,
   en:OrganisationIdentification.LEI ;
.
```

Details for a single business attribute, which defines a field representing a business identifier, are as follows:

```
en:OrganisationIdentification.LEI
 rdfs:label "OrganisationIdentification.LEI";
 a :BusinessAttribute;
 xmi:id "_yagFILl5EeOpCN0IL2Swqw";
 rdf:ID "_yagFILl5EeOpCN0IL2Swqw";
:BusinessAttribute\/name  "LEI" ;
:BusinessAttribute\/definition  "Legal Entity Identifier is a code allocated to a party as
described in ISO 17442 \"Financial Services - Legal Entity Identifier (LEI)\"." ;
:BusinessAttribute\/registrationStatus  "Registered" ;
:BusinessAttribute\/maxOccurs  1 ;
:BusinessAttribute\/minOccurs  1 ;
:BusinessAttribute\/isDerived  false ;
:BusinessAttribute\/derivation en:PersonOrOrganisation1Choice.LEI,
en:PersonOrOrganisation2Choice.LEI , en:SecuritiesTransactionTransmission2.
TransmittingBuyer , en:SecuritiesTransactionTransmission2.TransmittingSeller
, en:DerivativePartyIdentification1Choice.LEI , en:FinancialInstrument53.LEI,
en:FinancialInstrument48Choice.LEI, en:SecuritiesReferenceDataReport6.Issuer ;
:BusinessAttribute\/simpleType en:LEIIdentifier ;
```

The last property, linking to `en:LEIIdentifier`, links to the following `:IdentifierSet` object. The `:IdentifierSet` adds details how the LEI is represented as a string, particularly, it contains a regular expression in the `:IdentifierSet\/pattern` property which constrains the format of the LEI:

```
en:LEIIdentifier
 rdfs:label "LEIIdentifier";
 a :IdentifierSet;
 rdfs:seeAlso i:LEIIdentifier ;
 xmi:id "_h7Yn9yjAEeKnA5P_jl2DUw";
 rdf:ID "_h7Yn9yjAEeKnA5P_jl2DUw";
:IdentifierSet\/name  "LEIIdentifier" ;
:IdentifierSet\/definition  "Legal Entity Identifier is a code allocated to a party as
described in ISO 17442 \"Financial Services - Legal Entity Identifier (LEI)
\"." ; :IdentifierSet\/registrationStatus  "Registered" ;
:IdentifierSet\/pattern  "[A-Z0-9]{18,18}[0-9]{2,2}" ;
:IdentifierSet\/identificationScheme  "Global LEI System; LEIIdentifier" ;
:IdentifierSet\/example "123456789012345678" ;
```

In the e-Repository, a pair of `:BusinessAssociationEnd`(s) plays a role similar to a pair of `owl:ObjectProperty`(s) related by `owl:inverseOf`. The following example links `en:OrganisationName` to `en:OrganisationIdentification`:

```
en:OrganisationIdentification.OrganisationName
 rdfs:label "OrganisationIdentification.OrganisationName";
 a :BusinessAssociationEnd;
 xmi:id "_wd0NhA93EeGeV5vP7Mvdig_-1425645576";
 rdf:ID "_wd0NhA93EeGeV5vP7Mvdig_-1425645576";
:BusinessAssociationEnd\/name  "OrganisationName" ;
:BusinessAssociationEnd\/definition  "Name by which an organisation is known and which is
usually used to identify that organisation. It is derived from the association between
PartyIdentificationInformation and PartyName." ;
:BusinessAssociationEnd\/registrationStatus  "Registered" ;
:BusinessAssociationEnd\/minOccurs  0 ;
:BusinessAssociationEnd\/isDerived  false ;
:BusinessAssociationEnd\/opposite en:OrganisationName.Organisation ;
:BusinessAssociationEnd\/type en:OrganisationName ;
.
en:OrganisationName.Organisation
 rdfs:label "OrganisationName.Organisation";
 a :BusinessAssociationEnd;
 xmi:id "___VtOWIiEeGD28DQaMef-g";
 rdf:ID "___VtOWIiEeGD28DQaMef-g";
:BusinessAssociationEnd\/name  "Organisation" ;
:BusinessAssociationEnd\/definition  "Organisation identification which contains a name." ;
:BusinessAssociationEnd\/registrationStatus  "Registered" ;
:BusinessAssociationEnd\/maxOccurs  1 ;
:BusinessAssociationEnd\/minOccurs  0 ;
:BusinessAssociationEnd\/isDerived  false ;
:BusinessAssociationEnd\/opposite en:OrganisationIdentification.OrganisationName ;
:BusinessAssociationEnd\/type en:OrganisationIdentification ;
```

In many systems a name is simply a literal string, but that is not the case for `en:OrganisationName`. `en:OrganisationName` is a composite object (a business component) that can contain multiple literal strings representing different names such as a legal name, trading name or a short name.

```
en:OrganisationName
 rdfs:label "OrganisationName";
 a :BusinessComponent;
 rdfs:seeAlso i:OrganisationName ;
 xmi:id "_E7L5pcTGEeChad0JzLk7QA_255281013";
 rdf:ID "_E7L5pcTGEeChad0JzLk7QA_255281013";
:BusinessComponent\/name  "OrganisationName" ;
:BusinessComponent\/definition  "Name by which an organisation is known and which is usually
used to identify that organisation." ;
:BusinessComponent\/registrationStatus  "Registered" ;
:BusinessComponent\/superType en:PartyName ;
:BusinessComponent\/associationDomain en:OrganisationIdentification.OrganisationName ;
:BusinessComponent\/element
   en:OrganisationName.Organisation ,
   en:OrganisationName.LegalName ,
   en:OrganisationName.TradingName ,
   en:OrganisationName.ShortName ;
.
en:OrganisationName.Organisation
 rdfs:label "OrganisationName.Organisation";
 a :BusinessAssociationEnd;
 xmi:id "___VtOWIiEeGD28DQaMef-g";
 rdf:ID "___VtOWIiEeGD28DQaMef-g";
:BusinessAssociationEnd\/name  "Organisation" ;
:BusinessAssociationEnd\/definition  "Organisation identification which contains a name." ;
:BusinessAssociationEnd\/registrationStatus  "Registered" ;
:BusinessAssociationEnd\/maxOccurs  1 ;
:BusinessAssociationEnd\/minOccurs  0 ;
:BusinessAssociationEnd\/isDerived  false ;
:BusinessAssociationEnd\/opposite en:OrganisationIdentification.OrganisationName ;
:BusinessAssociationEnd\/type en:OrganisationIdentification ;
.
```

```
en:OrganisationName.LegalName
 rdfs:label "OrganisationName.LegalName";
 a :BusinessAttribute;
 xmi:id "_E7L5psTGEeChad0JzLk7QA_-805921363";
 rdf:ID "_E7L5psTGEeChad0JzLk7QA_-805921363";
:BusinessAttribute\/name  "LegalName" ;
:BusinessAttribute\/definition  "Official name under which an organisation is registered." ;
:BusinessAttribute\/registrationStatus  "Registered" ;
:BusinessAttribute\/maxOccurs  1 ;
:BusinessAttribute\/minOccurs  1 ;
:BusinessAttribute\/isDerived  false ;
:BusinessAttribute\/simpleType en:Max35Text ;
.
en:OrganisationName.TradingName
 rdfs:label "OrganisationName.TradingName";
 a :BusinessAttribute;
 xmi:id "_E7L5qMTGEeChad0JzLk7QA_1729897932";
 rdf:ID "_E7L5qMTGEeChad0JzLk7QA_1729897932";
:BusinessAttribute\/name  "TradingName" ;
:BusinessAttribute\/definition  "Name used by a business for commercial purposes, although its
registered legal name, used for contracts and other formal situations, may be another." ;
:BusinessAttribute\/registrationStatus  "Registered" ;
:BusinessAttribute\/maxOccurs  1 ;
:BusinessAttribute\/minOccurs  1 ;
:BusinessAttribute\/isDerived  false ;
:BusinessAttribute\/simpleType en:Max350Text ;
.
```

8.3 Message components and attributes

Messages are described by a schema in the e-Repository written in terms of types such as `:MessageDefinition`, `:MessageComponent`, `:MessageAttribute`, etc . This clause starts at the root of the definition of the auth.016 message and works down to a single string-valued type. The uppermost type in the definition is `:MessageDefinition`:

```
en:FinancialInstrumentReportingTransactionReportV01
 rdfs:label "FinancialInstrumentReportingTransactionReportV01";
 a :MessageDefinition;
 rdfs:seeAlso i:FinancialInstrumentReportingTransactionReportV01 ;
 xmi:id "_4LUGT0RNEee7JdgA9zPESA";
 rdf:ID "_4LUGT0RNEee7JdgA9zPESA";
:MessageDefinition\/name  "FinancialInstrumentReportingTransactionReportV01" ;
:MessageDefinition\/definition  "The FinancialInstrumentReportingTransactionReport message
is sent by the reporting agent to the competent authority to report on the securities
transactions or by the competent authority to another competent authority when the transaction
needs to be exchanged between the competent authorities." ;
:MessageDefinition\/registrationStatus  "Registered" ;
:MessageDefinition\/messageSet
en:Financial%20Instruments%20and%20Transactions%20Regulatory%20Reporting%20%28Transactions%20
and%20Financial%20Instruments%20Data%20Reporting%29%20-%20ISO%20-%20Latest%20version ;
:MessageDefinition\/xmlTag  "FinInstrmRptgTxRpt" ;
:MessageDefinition\/rootElement  "Document" ;
:MessageDefinition\/messageBuildingBlock
  en:FinancialInstrumentReportingTransactionReportV01.Transaction ,
  en:FinancialInstrumentReportingTransactionReportV01.SupplementaryData ;
:MessageDefinition\/messageDefinitionIdentifier
en:FinancialInstrumentReportingTransactionReportV01.id ;
.
```

A `:MessageDefinitionIdentifier` represents the fact that this is called the “auth.016” message in a structured way:

```
en:FinancialInstrumentReportingTransactionReportV01.id
      a :MessageDefinitionIdentifier;
:MessageDefinitionIdentifier\/businessArea  "auth" ;
:MessageDefinitionIdentifier\/messageFunctionality  "016" ;
:MessageDefinitionIdentifier\/flavour  "001" ;
:MessageDefinitionIdentifier\/version  "01" ;
      .
```

The :MessageBuildingBlock is one of a family of objects that define the structure of a message in a manner similar to the XML schema (XSD), closely enough that an XSD schema can be derived from the e-Repository:

```
en:FinancialInstrumentReportingTransactionReportV01.Transaction
 rdfs:label "FinancialInstrumentReportingTransactionReportV01.Transaction";
 a :MessageBuildingBlock;
 xmi:id "_4LUGUURNEee7JdgA9zPESA";
 rdf:ID "_4LUGUURNEee7JdgA9zPESA";
:MessageBuildingBlock\/name  "Transaction" ;
:MessageBuildingBlock\/definition  "Provides the details of the reported securities
transactions." ;
:MessageBuildingBlock\/registrationStatus  "Provisionally Registered" ;
:MessageBuildingBlock\/minOccurs  1 ;
:MessageBuildingBlock\/xmlTag  "Tx" ;
:MessageBuildingBlock\/complexType en:ReportingTransactionType1Choice ;
.
```

The above building block contains a `:ChoiceComponent`, which represents an `<xsd:choice>` element in an XSD schema, which allows a choice of one of many elements to appear in a position:

```
en:ReportingTransactionType1Choice
 rdfs:label "ReportingTransactionType1Choice";
 a :ChoiceComponent;
 xmi:id "_JzZ68YG-EeaalK9UbuVGFw";
 rdf:ID "_JzZ68YG-EeaalK9UbuVGFw";
:ChoiceComponent\/name  "ReportingTransactionType1Choice" ;
:ChoiceComponent\/definition  "Choice between a new or a cancellation transaction." ;
:ChoiceComponent\/registrationStatus  "Registered" ;
:ChoiceComponent\/messageBuildingBlock
en:FinancialInstrumentReportingTransactionReportV01.Transaction ;
:ChoiceComponent\/messageElement
  en:ReportingTransactionType1Choice.New ,
  en:ReportingTransactionType1Choice.Cancellation ,
  en:ReportingTransactionType1Choice.SupplementaryData ;
.
```

The most important relationship in the above is `:ChoiceComponent\/messageElement` which links to three choices. A `<New>` element appears here in the XML document if the auth.016 document was a new regulatory filing: in either the MOF or RDF (the G_5 graph) `en:ReportingTransactionType1Choice.New` is an object property of the `en:ReportingTransactionType1Choice` node. (The link between the G_5 graph and the M_c graph is that the G_5 graph uses the types and predicates that are referred to in the M_c graph.) As with the business components, this fact is represented by a single `:MessageAssociationEnd` as `en:ReportingTransactionType1Choice.New` does not have an inverse property as the graph is structured as a tree without cycles: `:MessageAssociationEnd(s)`:

```
en:ReportingTransactionType1Choice.New
 rdfs:label "ReportingTransactionType1Choice.New";
 a :MessageAssociationEnd;
 xmi:id "_J-kCEYG-EeaalK9UbuVGFw";
 rdf:ID "_J-kCEYG-EeaalK9UbuVGFw";
:MessageAssociationEnd\/name  "New" ;
:MessageAssociationEnd\/definition  "Transaction is a newly reported transaction." ;
:MessageAssociationEnd\/registrationStatus  "Provisionally Registered" ;
:MessageAssociationEnd\/maxOccurs  1 ;
:MessageAssociationEnd\/minOccurs  1 ;
:MessageAssociationEnd\/xmlTag  "New" ;
:MessageAssociationEnd\/businessComponentTrace en:SecuritiesTransaction ;
:MessageAssociationEnd\/isDerived  false ;
:MessageAssociationEnd\/isComposite  true ;
:MessageAssociationEnd\/type en:SecuritiesTransactionReport4 ;
.
```

The `:MessageAssociationEnd\/type` property connects an `<xsd:choice>` in the XSD schema to a `:MessageComponent`, which is in this case an object that represents a security transaction:

```
en:SecuritiesTransactionReport4
 rdfs:label "SecuritiesTransactionReport4";
 a :MessageComponent;
 rdfs:seeAlso i:SecuritiesTransactionReport4 ;
 xmi:id "_J-kCF4G-EeaalK9UbuVGFw";
 rdf:ID "_J-kCF4G-EeaalK9UbuVGFw";
:MessageComponent\/name  "SecuritiesTransactionReport4" ;
:MessageComponent\/definition  "Details of the securities transaction report." ;
:MessageComponent\/registrationStatus  "Registered" ;
:MessageComponent\/trace en:SecuritiesTransaction ;
:MessageComponent\/messageElement
  en:SecuritiesTransactionReport4.TransactionIdentification ,
  en:SecuritiesTransactionReport4.ExecutingParty ,
  en:SecuritiesTransactionReport4.InvestmentPartyIndicator ,
  en:SecuritiesTransactionReport4.SubmittingParty ,
  en:SecuritiesTransactionReport4.Buyer ,
  en:SecuritiesTransactionReport4.Seller ,
  en:SecuritiesTransactionReport4.OrderTransmission ,
  en:SecuritiesTransactionReport4.Transaction ,
  en:SecuritiesTransactionReport4.FinancialInstrument ,
  en:SecuritiesTransactionReport4.InvestmentDecisionPerson ,
  en:SecuritiesTransactionReport4.ExecutingPerson ,
  en:SecuritiesTransactionReport4.AdditionalAttributes ,
  en:SecuritiesTransactionReport4.TechnicalAttributes ,
  en:SecuritiesTransactionReport4.SupplementaryData ;
.
```

One very simple attribute of the transaction is the `en:SecuritiesTransactionReport4.TransactionIdentification`, which is just a string which uniquely identifies the transaction:

```
en:SecuritiesTransactionReport4.TransactionIdentification
 rdfs:label "SecuritiesTransactionReport4.TransactionIdentification";
 a :MessageAttribute;
 xmi:id "_KJ36MYG-EeaalK9UbuVGFw";
 rdf:ID "_KJ36MYG-EeaalK9UbuVGFw";
:MessageAttribute\/name  "TransactionIdentification" ;
:MessageAttribute\/definition  "Unique and unambiguous identification of the transaction." ;
:MessageAttribute\/registrationStatus  "Provisionally Registered" ;
:MessageAttribute\/maxOccurs  1 ;
:MessageAttribute\/minOccurs  1 ;
:MessageAttribute\/xmlTag  "TxId" ;
:MessageAttribute\/businessElementTrace en:TradeIdentification.Identification ;
:MessageAttribute\/isDerived  false ;
:MessageAttribute\/simpleType en:Max52Text ;
.
```

8.4 CodeSets

8.4.1 General

A CodeSet is a “complete and enumerated set of codes grouped together to characterize all the values of an attribute” (see ISO 20022-1:2013, 3.24). A common pattern in the ISO 20022 e-repository is for a versioned CodeSets that types a MessageElement to trace from an unversioned CodeSet that types the corresponding BusinessElement traced from the MessageElement. CodeSets can also be unrelated to other codes and can even be empty.

8.4.2 Empty CodeSet

Mechanical conversion to Turtle of an empty CodeSet shows its identification, labelling, description and related attributes. In the following example, it appears constraint was not fully implemented in the conversion to named nodes as there is not a constraint that the names of constraint must be unique:

```
en:ActiveCurrencyCode
 rdfs:label "ActiveCurrencyCode";
 a :CodeSet;
 rdfs:seeAlso i:ActiveCurrencyCode ;
 xmi:id "_bqIp5tp-Ed-ak6NoX_4Aeg_-1326801359";
 rdf:ID "_bqIp5tp-Ed-ak6NoX_4Aeg_-1326801359";
:CodeSet\/name  "ActiveCurrencyCode" ;
:CodeSet\/definition  "A code allocated to a currency by a Maintenance Agency under an
international identification scheme as described in the latest edition of the international
standard ISO 4217 \"Codes for the representation of currencies and funds\"." ;
:CodeSet\/registrationStatus  "Registered" ;
:CodeSet\/pattern  "[A-Z]{3,3}" ;
:CodeSet\/trace en:ParentCurrencyCode ;
:CodeSet\/example "EUR" ;
:CodeSet\/constraint en:ActiveCurrency ;
.
```

8.4.3 CodeSet with codes

As with an empty CodeSet, its attributes are listed, but this time with the addition of a node per code. In this example, the codes in the unversioned CodeSet have information about the codeName and definition that is missing from the codes in the derived versioned CodeSet. The codeNames are inferred at time of generating XML schema from the versioned CodeSet by referencing the Code in the unversioned CodeSet with the matching code name, rather than by direct reference to the traced node.

```
en:DistributionPolicyCode
 rdfs:label "DistributionPolicyCode";
 a :CodeSet;
 rdfs:seeAlso i:DistributionPolicyCode ;
 xmi:id "_awwx5dp-Ed-ak6NoX_4Aeg_-1525096597";
 rdf:ID "_awwx5dp-Ed-ak6NoX_4Aeg_-1525096597";
:CodeSet\/name  "DistributionPolicyCode" ;
:CodeSet\/definition  "Specifies if income is to be paid out (distributed) or retained
(accumulated)." ;
:CodeSet\/registrationStatus  "Registered" ;
:CodeSet\/derivation "_awwx4tp-Ed-ak6NoX_4Aeg_-433302197" ;
:CodeSet\/example "DIST" ;
:CodeSet\/code
  en:DistributionPolicyCode.Distribution ,
  en:DistributionPolicyCode.Accumulation ;
.
en:DistributionPolicyCode.Distribution
 rdfs:label "DistributionPolicyCode.Distribution";
 a :Code;
 xmi:id "_awwx5tp-Ed-ak6NoX_4Aeg_1041617774";
 rdf:ID "_awwx5tp-Ed-ak6NoX_4Aeg_1041617774";
:Code\/name  "Distribution" ;
:Code\/definition  "Income is distributed to the investors in the fund." ;
:Code\/registrationStatus  "Registered" ;
:Code\/codeName  "DIST" ;
.
en:DistributionPolicyCode.Accumulation
 rdfs:label "DistributionPolicyCode.Accumulation";
 a :Code;
 xmi:id "_aw6i4Np-Ed-ak6NoX_4Aeg_1041617805";
 rdf:ID "_aw6i4Np-Ed-ak6NoX_4Aeg_1041617805";
:Code\/name  "Accumulation" ;
:Code\/definition  "Income is added to the capital of the fund." ;
:Code\/registrationStatus  "Registered" ;
:Code\/codeName  "ACCU" ;
.

en:DistributionPolicy1Code
 rdfs:label "DistributionPolicy1Code";
 a :CodeSet;
 rdfs:seeAlso i:DistributionPolicy1Code ;
 xmi:id "_awwx4tp-Ed-ak6NoX_4Aeg_-433302197";
 rdf:ID "_awwx4tp-Ed-ak6NoX_4Aeg_-433302197";
:CodeSet\/name  "DistributionPolicy1Code" ;
:CodeSet\/definition  "Specifies if income is to be paid out (distributed) or retained
(accumulated)." ;
:CodeSet\/registrationStatus  "Registered" ;
:CodeSet\/trace "_awwx5dp-Ed-ak6NoX_4Aeg_-1525096597" ;
:CodeSet\/example "DIST" ;
:CodeSet\/code
  en:DistributionPolicy1Code.Distribution ,
  en:DistributionPolicy1Code.Accumulation ;
.
en:DistributionPolicy1Code.Distribution
 rdfs:label "DistributionPolicy1Code.Distribution";
 a :Code;
 xmi:id "_awwx49p-Ed-ak6NoX_4Aeg_-433302196";
 rdf:ID "_awwx49p-Ed-ak6NoX_4Aeg_-433302196";
:Code\/name  "Distribution" ;
:Code\/registrationStatus  "Provisionally Registered" ;
.
en:DistributionPolicy1Code.Accumulation
 rdfs:label "DistributionPolicy1Code.Accumulation";
 a :Code;
 xmi:id "_awwx5Np-Ed-ak6NoX_4Aeg_-433302187";
 rdf:ID "_awwx5Np-Ed-ak6NoX_4Aeg_-433302187";
:Code\/name  "Accumulation" ;
:Code\/registrationStatus  "Provisionally Registered" ;
.
```

8.5 Coining unique identifiers

8.5.1 The problem

RDFS and OWL assume that a predicate means the same thing wherever it appears in the graph, so to support standard inference, it is necessary to assign a uniquely named predicate for each meaning a property has in the graph. If any disambiguation has to be done, it has to be done when creating the inference-ready G_5 graph. This subclause describes several possible strategies for doing so.

8.5.2 Composite names

The strategy used in this subclause is to generate identifiers for schema terms by concatenating multiple names from the source schema to a namespace URI. For instance, the names of attributes in the e-Repository are not unique, but they can be made unique by concatenating them to the name of the types that they act on.

This strategy is like the use of multipart primary keys in a relational database and, like the relational database, requires determining what the key structure is for a given system.

8.5.3 Arbitrary disambiguators

People like having human readable identifiers: the composite names used in this subclause are meaningful to people because they group together the attributes shared by a given type. As comfortable as they are, this make it more and more difficult to maintain a systematic naming scheme as the number of objects increases.

Lacking a strategy to make meaningful names, one approach to disambiguating is to add arbitrary information, such as a number, to otherwise non-unique names. This strategy is used in the e-Repository, which contains names such as `FinancialInstrument53` and `SecuritiesReferenceDataReport6`. This puts a burden on the user, who needs to know which kind of financial instrument `FinancialInstrument53` is or what number they are supposed to append to the financial instrument that they have in mind. As with the other naming strategies discussed in this document, better tools can help.

In addition, arbitrary disambiguators create operational problems because they are arbitrary: if the process of building a graph is begun from scratch, it is possible that different numbers are assigned. Worse yet, if the development of vocabulary was pursued in two different forks there is the risk that the same name can be applied to two different concepts in the two forks.

8.5.4 External id

It turns out that the XMI file of the e-Repository contains unique identifiers for objects (the xmi:id) that are opaque to humans. One choice is to derive URIs from those identifiers. This choice was made in the following file associated with this document:

`Financial_Instruments_and_Transactions_Regulatory_Reporting.iso20022+xmi-id.RDF.TTL`

An example of a business attribute written in this vocabulary is:

```
x:_yagFILl5EeOpCN0IL2Swqw
 rdfs:label "OrganisationIdentification.LEI";
 a :BusinessAttribute;
 xmi:id "_yagFILl5EeOpCN0IL2Swqw";
 rdf:ID "_yagFILl5EeOpCN0IL2Swqw";
:BusinessAttribute\/name  "LEI" ;
:BusinessAttribute\/definition  "Legal Entity Identifier is a code allocated to a party as
described in ISO 17442 \"Financial Services - Legal Entity Identifier (LEI)\"." ;
:BusinessAttribute\/registrationStatus  "Registered" ;
:BusinessAttribute\/maxOccurs  1 ;
:BusinessAttribute\/minOccurs  1 ;
:BusinessAttribute\/isDerived  false ;
:BusinessAttribute\/derivation x:_gOV-kgkiEeWGouz230Xp5Q, x:_-GX-QQkiEeWGouz230Xp5Q,
x:_9VEUEeqMEeSsk6KxwbYJ9w, x:_1GQfkOqMEeSsk6KxwbYJ9w, x:_unMS4EW8EeWaZZ6gWK8UVw ,
x:_a46r4vBqEeWTAY6i--T_aA, x:_HYrJU35aEea2k7EBUopqxw, x:_LGQmKSX7EeigZbhgJcrASA ;
:BusinessAttribute\/simpleType x:_h7Yn9yjAEeKnA5P_jl2DUw ;
.
```

This vocabulary has the critical property of being unique but is much more difficult than previous vocabularies to understand manually.

8.5.5 Internal id

Some programming languages such as Python with the id() function can generate a unique identifier based on memory addresses. When working with a collection of objects created by a library such as pyecore, the uniqueness of the identifiers inside a single run of a program can be counted on and used to generate an RDF graph with consistent relationships. Like the external identifiers, the identifiers can be confusing to users and there is the additional disadvantage that the identifiers will not be stable from one run of the program to the next, making the conversion process irreproducible.

8.5.6 Random id

A randomly generated identifier such as the version 4 UUID (see RFC 4122) supports the creation of an astronomical number of unique identifiers by independent parties without communication. If these parties later share graphs they have created, they can be sure the probability of a conflict is vanishingly small.

Applied directly, random identifiers are not stable from one run to the next, but this can be remedied, as with the arbitrary disambiguators, by making a database of object-to-id assignments that can be used to look up the name previously used for an object.

8.5.7 Skolemization

An approach to unique id generation similar to the use of composite keys is to create a composite key, but instead of concatenating it to a namespace, to pass that composite key through a hash function (e.g. SHA-256) to produce an identifier that looks like a random id. This process is similar to the process used to create a Version 5 UUID (see RFC 4122) and has the advantage (and disadvantage) of being opaque to end users and resistant to manipulation.

9 Representing the ISO 20022 e-Repository in OWL and RDFS

9.1 Methodology

This clause concerns the conversion of the e-Repository into OWL and RDFS vocabulary. Unlike the work in Clause 8, this work does not preserve the original context exactly, but rather tries to express as much meaning as possible with OWL and RDFS. This conversion process uses the same method to coin names for object such as `en:Asset` as used in Clause 8, but the predicates and types used in the schema come from OWL and RDFS.

This clause illustrates a representation of the e-Repository in OWL based on the G_4 graph that represents some part of messages as properties and others as classes, with the result that the ontology defines hundreds of properties. Annex A shows a representation, under development, that consistently uses classes to represent message parts and uses a small library of properties built into a foundation ontology to interconnect them.

OWL inference tools such as Protégé typically require ontologies to conform to OWL DL, which requires a clear separation between ontology objects (definitions of properties and classes) and instance objects (instances of classes defined in the ontology.) Instance objects can be given properties that are defined in the ontology (which fully participate in inference), but ontology objects can only be given properties that are defined in the OWL and RDFS specifications or that are annotation properties defined in the ontology that are excluded from inference.

Figure 2 shows the structure of a valid OWL DL ontology that completely captures the content of the e-Repository.

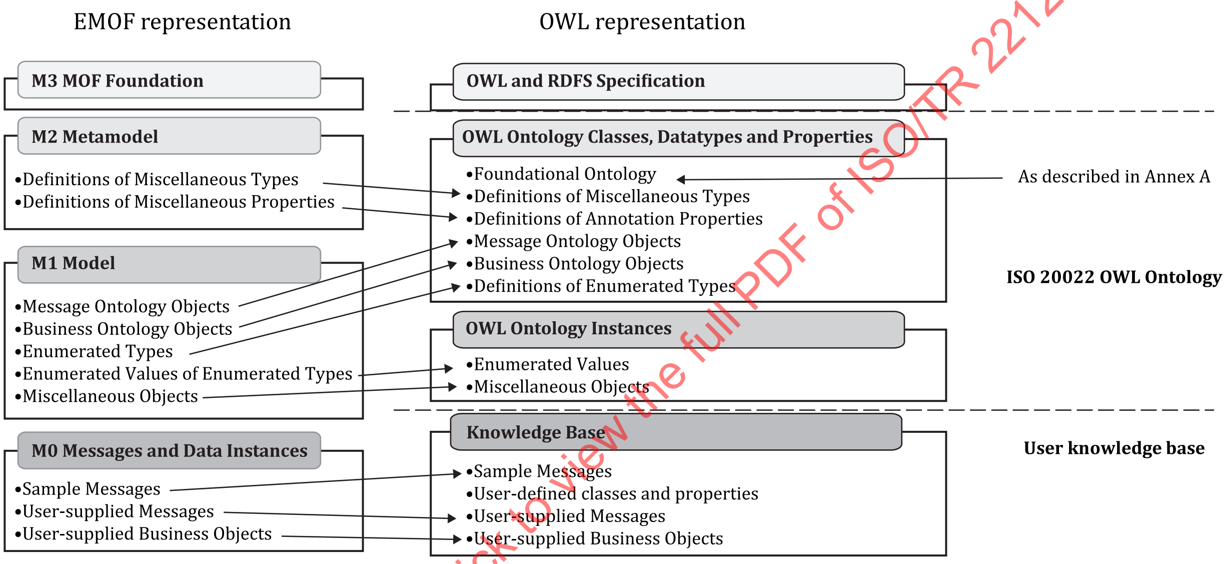

EMOF representation
OWL representation
M3 MOF Foundation
OWL and RDFS Specification
M2 Metamodel
•Definitions of Miscellaneous Types
•Definitions of Miscellaneous Properties
OWL Ontology Classes, Datatypes and Properties
•Foundational Ontology
•Definitions of Miscellaneous Types
•Definitions of Annotation Properties
•Message Ontology Objects
•Business Ontology Objects
•Definitions of Enumerated Types
As described in Annex A
ISO 20022 OWL Ontology
M1 Model
•Message Ontology Objects
•Business Ontology Objects
•Enumerated Types
•Enumerated Values of Enumerated Types
•Miscellaneous Objects
OWL Ontology Instances
•Enumerated Values
•Miscellaneous Objects
M0 Messages and Data Instances
•Sample Messages
•User-supplied Messages
•User-supplied Business Objects
Knowledge Base
•Sample Messages
•User-defined classes and properties
•User-supplied Messages
•User-supplied Business Objects
User knowledge base

Figure 2 — Mapping of EMOF layers to OWL DL

Ultimately the metamodel (M2 level) is defined using foundational objects defined at the M3 level of EMOF. These foundational definitions are not transcribed to the OWL ontology. Although these are the underlying definitions of the system, they play a different role than the definitions of the OWL specification because, for the most part, properties and classes defined in the OWL specification play the same role as definitions at the M2 level when M1 level objects are described in the OWL ontology.

The OWL ontology can contain foundational definitions that do not exist in EMOF. Such definitions are not used in this clause, but such definitions are developed in the work described in Annex A.

Most of the content translated to OWL comes from the M1 (model) level of EMOF. Many objects at the M1 level are ontology objects that map naturally to ontology objects in OWL. Although these objects are described with objects defined at the M2 (metamodel) level of EMOF, these are described using standard OWL terminology and the corresponding definitions from the M2 level are not imported from OWL.

The metamodel is a richer than OWL standard terminology in some respects because it distinguishes between different types of objects and properties (such as `:BusinessAssociationEnd`, `:MessageComponent`, `:BusinessComponent`) and defines properties not defined in OWL. OWL DL requires that a class be always defined as an `owl:Class`, and a property as one of three types, but special kinds of class and property are designated by making them subclasses or subproperties of another class. Properties of ontology objects that do not exist in the OWL specification are captured as annotation properties defined in the OWL specification.

Some objects at the M1 level correspond to instance objects as opposed to ontology objects in OWL, and for these objects and their properties, definitions from the M2 level are represented as ontology objects in OWL.

Enumerated values are an important class of objects from the M1 level that are represented as instance objects in the OWL ontology. OWL offers the choice of representing enumeration values as either instance objects or datatypes. Subclause 9.4 demonstrates the representation of enumeration-valued properties as object properties, where properties of the enumerated values themselves can be represented as datatype, object or annotation properties as defined in the ontology. An alternate approach is to represent enumeration-valued properties as datatype properties and define restriction types such that the presence of a certain enumerated value implies that an object is instance of a particular subtype. In that case, annotation properties can be used to describe the class of objects to which objects with a particular annotation value apply.

9.2 RDFS modelling of a business component

`en:Asset` is a business component can be used to demonstrate modelling with the simple RDFS schema language.

The metamodel contains a number of concepts such as message building block, business component, message component, message attribute and business association ends that are mapped to two concepts in RDFS: `rdfs:Class` or `rdfs:Property`.

`en:Asset` is declared as a class, which has a language-tagged label and description derived from the e-Repository:

```
en:Asset
 a rdfs:Class ;
 rdfs:subClassOf:BusinessComponent ;
 rdfs:label "Asset"@en ;
 rdfs:comment "Tangible items of value to a business."@en ;
.
```

`en:Asset.ExpiryDate` is a business attribute in EMOF and is represented as a `rdf:Property`. Both properties are linked to `en:Asset` through the `rdfs:domain` property, whereas the `rdfs:range` points to a user-defined literal type `en:ISODateTime`:

```
en:Asset.ExpiryDate
 a rdf:Property ;
 rdfs:subClass en:BusinessAttribute;
 rdfs:label "ExpiryDate" ;
 rdfs:comment "Date on which an order, a privilege, an entitlement or an offer
ates.
For an interest bearing asset, it is the date/time at which it becomes due and has to
 be repaid." ;
 rdfs:domain en:Asset ;
 rdfs:range en:ISODateTime .
```

`en:Asset.Derivative` is a business association end in EMOF which is represented as an `rdf:Property` here:

```
en:Asset.Derivative
 a rdf:Property ;
 rdfs:subPropertyOf en:BusinessAssociationEnd ;
 rdfs:label "Derivative" ;
 rdfs:comment "Specifies the parameters of a derivative instrument based on a specific
 asset."@en ;
 rdfs:domain en:Asset ;
 rdfs:range en:Derivative .
```

Notably, RDF domain and range properties are less specific than the OWL restrictions defined in 9.3 and would not be added to an OWL ontology together with restrictions.

9.3 Datatype and object properties in OWL

OWL takes a different approach to describing properties from RDFS. RDFS cannot, for instance, specify that a message must contain exactly one trend identifier, but OWL can. Any property must be specified as either a datatype or object property.

```
en:SecuritiesTransactionReport4.TransactionIdentification
     a owl:DatatypeProperty ;
     rdfs:subPropertyOf en:MessageAttribute ;
     rdfs:label “Transaction Identification”@en
     rdfs:comment "Unique and unambiguous identification of the transaction."@en ;
```

Properties are connected to classes by setting restrictions:

```
en:SecuritiesTransactionReport4
    a owl:Class ;
    rdfs:label "SecuritiesTransactionReport4" ;
    rdfs:comment "Details of the securities transaction report." ;
    rdfs:subClassOf [
        a owl:Restriction ;
        owl:onProperty
           en:SecuritiesTransactionReport4.TransactionIdentification;
        owl:allValuesFrom en:Max52Text.

] ;
rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty
       en:SecuritiesTransactionReport4.TransactionIdentification;
    owl:minCardinality 1 .
];
rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty
       en:SecuritiesTransactionReport4.TransactionIdentification;
    owl:maxCardinality 1 .
].
```

The same strategy is applied an object property. In this case, en:RegulatoryReport.Authority and en:RegulatoryAuthorityRole.RegulatoryReport are inverse properties of each other and are constrained by restrictions on both enclosing classes.

```
en:RegulatoryReport.Authority
    a owl:ObjectProperty ;
    rdfs:label "Authority" ;
    rdfs:comment " Entity requiring the regulatory reporting information." ;
    rdfs:subPropertyOf en:BusinessAssociationEnd
    owl:inverseOf en:RegulatoryAuthorityRole.RegulatoryReport .

en:RegulatoryAuthorityRole.RegulatoryReport
    a owl:ObjectProperty ;
    owl:inverseOf en:RegulatoryReport.Authority ;
    rdfs:label "RegulatoryReport" ;
    rdfs:comment "Report which was requested by the regulatory authority." .

en:RegulatoryReport
    a owl:Class ;
    rdfs:label "RegulatoryReport" ;
    rdfs:comment "Information needed due to regulatory and statutory requirements." ;
    rdfs:subClassOf en:BusinessComponent ;
    rdfs:subClassOf [
       a owl:Restriction ;
       owl:onProperty en:RegulatoryReport.Authority ;
       owl:allValuesFrom en:RegulatoryAuthorityRole
    ];
    rdfs:subClassOf [
       a owl:Restriction ;
       owl:onProperty en:RegulatoryReport.Authority ;
       owl:minCardinality 1 ;
    ] .

en:RegulatoryReportRole
    a owl:Class ;
    rdfs:label "RegulatoryAuthorityRole.RegulatoryReport";
    rdfs:comment "Report which was requested by the regulatory authority.";
    rdfs:subClassOf en:BusinessComponent ;
    rdfs:subClassOf [
       a owl:Restriction ;
       owl:onProperty en:RegulatoryReportRole.RegulatoryReport ;
       owl:allValuesFrom en:RegulatoryReport
    ];
    rdfs:subClassOf [
       a owl:Restriction ;
       owl:onProperty en:RegulatoryReportRole.RegulatoryReport ;
       owl:maxCardinality 1 ;
    ] .
```

9.4 Representing enumerated types

Many attributes in the ISO 20022 model are represented by a string in the XML document, such as "DACR". This is a datatype property in EMOF and can be represented as a datatype property in OWL. The same string can be used with different meanings in the document, and the instance conversion process can convert that string, in context, to a reference to an instance object, supplied with the ontology, that describes that value. A class is defined in the ontology, together with a number of ontology instances describing values that the enumeration can take.

```
en:DividendPolicyCode
   a owl:Class ;
   rdfs:subClassOf a :CodeSet ;
   rdfs:label "DividendPolicyCode"@en ;
   rdfs:comment "Specifies the dividend policy of the financial instrument." ;

   owl:oneOf (
      en:DividendPolicyCode.DailyAccruingDividend,
      en:DividendPolicyCode.Cash,
      en:DividendPolicyCode.Units,
      en:DividendPolicyCode.Both

 ) .

en:DividendPolicyCode.DailyAccruingDividend
   a :DividendPolicyCode ;
   rdfs:label " DailyAccruingDividend"@en ;
   rdfs:comment "Dividend is paid daily and can be accrued."@en ;
;
  :codeName "DACR" .

en:DividendPolicyCode.Cash
   a :DividendPolicyCode ;
   rdfs:label " Cash"@en ;
   rdfs:comment "Dividend is paid in cash."@en ;
;
  :codeName "CASH" .

en:DividendPolicyCode.Units
   a :DividendPolicyCode ;
   rdfs:label " Units"@en ;
   rdfs:comment "Dividend is paid in units."@en ;
;
  :codeName "UNIT" .

en:DividendPolicyCode.Both
   a :DividendPolicyCode ;
   rdfs:label " Both"@en ;
   rdfs:comment "Dividend is paid in both cash and units."@en ;
;
  :codeName "BOTH" .
```

```
[ a owl:AllDifferent;

  owl:distinctMembers (
     en.DividendPolicyCode.DailyAccruingDividend,
     en.DividendPolicyCode.Cash,
     en.DividendPolicyCode.Units,
     en.DividendPolicyCode.Both
  )
]
```

`:codeName` fulfils the essential function of representing the code name used in this field and can be represented as a property of `:CodeSet`. An instance of the `:CodeSet` in this ontology is a record describing a single value that can be held in any `CodeSet`. This is a by-product of the `:CodeSet` concept being imported from the M2 world of the metamodel to the M1 world of the model, in such a way that `en:DividendPolicyCode` is a subclass of `:CodeSet` as opposed to an instance. Such a property defined in the ontology has none of the restrictions that an annotation property has and can fully participate in logical inference.

9.5 Restrictions on data types

OWL offers facilities for restrictions on data types mirroring restrictions in XML schema and the e-Repository. Namely, the same restrictions that can be applied to datatypes in XML schema can be applied to datatypes in OWL. For instance:

```
en:Max35Text
 a rdfs:Datatype ;
 rdfs:label "Max35Text" ;
 rdfs:comment "Specifies a character string with a maximum length of 35 characters." ;
 owl:onDatatype xsd:string ;
 owl:withRestrictions (
  [ xsd:minLength 1 ]
  [ xsd:maxLength 35 ]
).
```

9.6 Disjoint classes

Unless otherwise specified, it is possible for an OWL instance to be an instance of any arbitrary set of classes. EMOF, on the other hand, supports Java style single inheritance at the M2 level at which the metamodel is defined. In a model like that, class relationships are highly constrained, and an instance can only be a member of a set of classes that form a specific chain of inheritance. In that case, it is intrinsic to the model that any set of sibling classes are disjoint to one another, a fact which can be expressed in OWL using the `owl:disjointWith` property.

At the M1 level where the model is defined, the metamodel provides a `BusinessComponent/subType` and `BusinessComponent/superType` properties which define single-inheritance relationships between business components. A set of siblings, such as the direct subtypes of `en:Asset` are mutually disjoint.

```
en:Security a owl:Class; rdfs:subClassOf en:Asset .
en:Derivative a owl:Class; rdfs:subClassOf en:Asset .
en:Money a owl:Class; rdfs:subClassOf en:Asset .
en:LetterOfCredit a owl:Class rdfs:subClassOf en:Asset .
en:Guarantee a owl:Class; rdfs:subClassOf en:Asset .
en:Commodity a owl:Class; rdfs:subClassOf en:Asset .

en:Security owl:disjointFrom en:Derivative, en:Money, en:LetterOfCredit, en:Guarantee,
en:Commodity.
en:Derivative owl:disjointFrom en:Money, en:LetterOfCredit, en:Guarantee, en:Commodity.
en:Money owl:disjointFrom en:LetterOfCredit, en:Guarantee, en:Commodity.
en:LetterOfCredit owl:disjointFrom en:Guarantee, en:Commodity.
en:Guarantere owl:disjointFrom en:Commodity.
```

The same is true for the five subclasses of `en:Security` and also for the 267 top level business components that lack an explicit supertype, so derive directly from `en:BusinessComponent` in this OWL ontology.

NOTE The number of `owl:disjointFrom` statements can get rather large since it takes N*(N-1)/2 statements to specify that N classes are disjoint.

The number of statements required to assert all disjoint relationships is reduced in size when disjoint relationships are reused across a hierarchy. At an upper level of the hierarchy, classes reflecting divisions of the model, such as :BusinessComponent, :MessageComponent and :ChoiceComponent are themselves disjoint and are disjoint from most classes introduced in the ontology such as classes that define sets of enumerated values.

9.7 Miscellaneous objects and annotation properties

Objects such as :MessageDefinitionIdentifier are not naturally modelled as classes or instances but can be modelled as instance objects in RDF:

```
en:FinancialInstrumentReportingStatusAdviceV01.id
 a :MessageDefinitionIdentifier ;
 rdfs:label "Financial instrument reporting status advice Version 01"@en ;
:MessageDefinitionIdentifier.businessArea "auth" ;
:MessageDefinitionIdentifier.messageFunctionality "031" ;
:MessageDefinitionIdentifier.flavour "001" ;
:MessageDefinitionIdentifier.version "01" ;
```

This class and the properties associated with it are defined by OWL classes and properties defined in the M2 metamodel brought together with the M1 model concepts such as en:Asset that are also represented as OWL classes and properties.

Additional data can be added to schema objects by the use of annotation properties. For instance:

```
en:Trade
 a owl:Class;
 rdfs:subClassOf:BusinessComponent;
 rdfs:label "Trade"@en ;
 rdfs:comment "Result of an order between at least two parties. A trade relates to the
delivery of goods and services, cash or securities.";
:BusinessComponent.registrationStatus "Registered";
```

Annotation properties need to be declared as such:

```
:BusinessComponent.registrationStatus
 a owl:AnnotationProperty.
```

OWL comes with five built in annotation properties (owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy) that follow the same rules as annotation properties defined in the ontology. OWL DL imposes the following restrictions on annotation properties:

a) Annotation properties are mutually distinct with object, datatype and ontology properties.

b) Annotation properties cannot be subject to property axioms, for instance, an annotation property cannot be an owl:subPropertyOf another property.

c) The object of an annotation property must be a data literal, IRI reference or an anonymous individual. [11][12]

Miscellaneous objects modelled as individuals with OWL 2 can be the subject of datatype and object properties defined in the ontology and can be exposed to the full power of inference. Annotation types, however, are necessary to model facts from the e-Repository about classes and properties which cannot be modelled with other OWL axioms.

9.8 Containment relationships and inference

The message components, attributes and associations are an ontology of messages and the parts of messages. Properties that are generated (in an instance) from XML elements and attributes such as en:SecuritiesTransaction1.TradeDate or en:SecuritiesTransaction1.Quantity all denote that that one part of a message is contained inside another part. This can be written in OWL with a number of statements such as:

```
:containsDirectly a owl:Object Property .
en:SecuritiesTransaction1.TradeDate owl:subPropertyOf:containsDirectly .
```

for all of the properties that represent the structure of the message.

Furthermore, a `:contains` property can be defined in the ontology such that:

```
:contains a owl:ObjectProperty, owl:TransitiveProperty;
   owl:subPropertyOf:containsDirectly .
```

With this definition, the `:contains` property links any part of a message instance to all the parts that it contains.

9.9 OWL inference and its limits

A major strength of OWL is the ability to define classes in terms of logical definitions, i.e. an ontology user can define new classes based on restrictions relevant to an application. For instance, there is the concept of an "odd lot" in trading which involves less than 100 shares:

```
:OddLotQuantity
   a rdfs:Datatype ;
   owl:equivalentClass [
      a rdfs:Datatype ;
      owl:onDatatype xsd:decimal ;
      owl:withRestrictions (
        [ xsd:maxExclusive "100"^xsd:decimal ]
      )
   ] .
:TradeQuantityNode
   a owl:Class ;

   rdfs:subPropertyOf en:SecurityQuantity;
   rdfs:subPropertyOf [
      a owl:Restriction ;
      owl:onProperty en:SecurityQuantity.Unit ;
      owl:allValuesFrom:OddLotQuantity
   ] .

:OddLotTrade
   a owl:Class;
   rdfs:subPropertyOf en:SecurityTrade ;
   rdfs:subPropertyOf [
      a owl:Restriction ;
      owl:onProperty en:SecurityTrade.TradeQuantity ;
      owl:allValuesFrom:TradeQuantityNode
   ].
```

Notably, OWL allows the definition of classes based on numeric comparison combined with other restrictions on properties that can be combined with union, intersection and complement operators.

OWL, however, lacks the ability to do other forms of mathematics. Specifically, another class that can be useful to define is a "large" trade which is worth more than a certain dollar amount, which involves multiplying the `:en.SecurityQuantity.Unit` with the `en:Price.Amount`, an operation that OWL cannot do.

Additional rule languages exist such as SPIN and SWRL that are packaged with common OWL ontology tools such as Protégé and TopBraid Composer[5] and, working closely with OWL, can extend the range of possible logical definitions.

[5] TopBraid Composer is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

10 Instances aligned to the e-Repository

10.1 The G_4 graph — Message expressed with message components

The G_4 graph is similar to the G_2 graph, but is written in terms aligned with the terms defined in the M_c schema and structures with `rdf:label` statements simplified to be interpretable by RDFS and OWL. Both of these codes are good codes:

```
@prefix: <urn:iso:std:iso:20022:tech:xsd:auth.016.001.01/> .
@prefix en: <https://iso20022.plus/semantic/repository/en/#> .
@prefix bare: <http://rdf.ontology2.com/bare-xml#> .
@prefix rdf: <https://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <https://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <https://www.w3.org/XML/1998/namespace> .
@prefix xsd: <https://www.w3.org/2001/XMLSchema#> .


[] a :Document ;
     en:FinancialInstrumentReportingTransactionReportV01 [
            en:FinancialInstrumentReportingTransactionReportV01.Transaction [
                    en:ReportingTransactionType1Choice.New [
                            en:SecuritiesTransactionReport4.AdditionalAttributes [
                                    en:SecuritiesTransactionIndicator2.
SecuritiesFinancingTransactionIndicator true;
                  ];
                            en:SecuritiesTransactionReport4.Buyer [
                                    en:PartyIdentification79.AccountOwner [
                                            en:PartyIdentification76.Identification [
                                                    en:PersonOrOrganisation1Choice.LEI
"11111111111111111111"^^xsd:string ] ] ] ;
                            en:SecuritiesTransactionReport4.ExecutingPerson [
                                    en:ExecutingParty1Choice.Algorithm "A123456"^^xsd:string ]
;
                            en:SecuritiesTransactionReport4.ExecutingParty
"00000000000000000000"^^xsd:string ;
                            en:SecuritiesTransactionReport4.FinancialInstrument [
                                    en:FinancialInstrumentAttributes3Choice.Identification
"AB1234567890"^^xsd:string ] ;

                            en:SecuritiesTransactionReport4.InvestmentPartyIndicator true ;
                            en:SecuritiesTransactionReport4.OrderTransmission [
                                    en:SecuritiesTransactionTransmission2.
TransmissionIndicator true ] ;
                            en:SecuritiesTransactionReport4.Seller [
                                    en:PartyIdentification79.AccountOwner [
                                            en:PartyIdentification76.Identification [
                                                    en:PersonOrOrganisation1Choice.LEI
"22222222222222222222"^^xsd:string ] ] ] ;
                            en:SecuritiesTransactionReport4.SubmittingParty
"00000000000000000000"^^xsd:string;
                            en:SecuritiesTransactionReport4.Transaction [
                                    en:SecuritiesTransaction1.Price [
                                            en:SecuritiesTransactionPrice4Choice.Price [
                                                    en:SecuritiesTransactionPrice2Choice.
MonetaryValue [
                                                            en:AmountAndDirection61.Amount
"13.5 EUR" ;
                                                                    ] ] ] ;
                                    en:SecuritiesTransaction1.Quantity [
                                            en:FinancialInstrumentQuantity25Choice.Unit
"100.0"^^xsd:decimal ] ;
                                    en:SecuritiesTransaction1.TradeDate
"2018-12-17T09:30:47+00:00"^^xsd:dateTime ;

                                    en:SecuritiesTransaction1.TradeVenue "XDUB"^^xsd:string ;
                                    en:SecuritiesTransaction1.TradingCapacity
"MTCH"^^xsd:string ] ;
                            en:SecuritiesTransactionReport4.TransactionIdentification
"T123456"^^xsd:string ] ] ] .
```

10.2 The G_5 graph — Abstract instance of a trade

Previous clauses of this document show graphs that represent the content of a message based on the schema of the message components from the e-Repository. This subclause concerns the G_5 graph, which captures most of the meaning of the auth.016 sample message using terms defined in the business components.

The document below was made by printing out the G_3 graph and manually mapping structures in the G_4 graph to corresponding structures written with terms from the B_C business component ontology. Notably, the G_5 graph is centred around a en:Trade object as an object in the world and makes no statement that this `en:Trade` exists in context to a report, a frame which is established by the message identifier in the G_5 graph.

The G_5 graph uses a small number of namespaces:

```
@prefix:     <urn:iso:std:iso:20022:2013:ecore#> .
@prefix en:   <https://iso20022.plus/semantic/repository/en/#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix this: <urn:iso:std:iso:20022:2013:extra#> .
```

Some message components, elements and attributes correspond directly to business components such as `en:TradePrice` or `en:SecurityQuantity`. Other concepts important in the message, such as the legal entity identifier, do not exist in the business components.

The message components for the auth.016 message allow several different identifier schemes to be used to identify parties such as the buyer and seller. The choice of identifier is specified by the choice of XML element that appears in the message. That XML element maps to a predicate in RDF, so the choice of a predicate determines the identifier scheme in use.

The business components work differently. The business components contain a concept, an `en:Scheme`, to represent an identifier scheme and defines a mechanism to associate an identifier which such a scheme. In addition to the `en:Trade` instance, the G_5 graph contains two schemes. What is defined in the schema in the message components (the M1 layer of the meta-object facility) is defined as instances (the M0 layer) of the business components:

```
this:LEI
    a                   en:Scheme ;
    en:Scheme.NameShort "LEI"@en ;
    en:Scheme.NameLong  "Legal Entity Identifier"@en ;
    en:Scheme.Description
                        "The Legal Entity Identifier (LEI) is a unique global
identifier for legal entities participating in financial transactions." .
this:MIC
    a                   en:Scheme;
en:Scheme.NameShort "MIC"@en ;
en:Scheme.NameLong "Market Identification Code" @en ;
en:Scheme.Description "The Market Identifier Code (MIC) (ISO 10383) is a unique
identification code used to identify securities trading exchanges, regulated and
non-regulated trading markets." .
```

Another structural difference between the message components and the business components (as realized in RDF) is that the message components use properties to distinguish the relationship between a party and a trade where the business components use a chain of two properties `en:SecuritiesTradePartyRole` and `en:Role.Player` to associate a party with a trade. The type of the intermediate object, an `en:Role`, determines the role this party plays in the trade:

```
 [
   a en:Trade, en:SecurityTrade, en:SecuritiesTransaction ;
    en:SecuritiesTradePartyRole [
        a en:BuyerRole ;
        en:Role.Player [
            a en:Party ;
            en:Party.Identification [
                a en:PartyIdentificationInformation ;
                en:PartyIdentificationInformation.OtherIdentification [
                    a en:GenericIdentification ;
                    en:GenericIdentification.Identification "11111111111111111111"^^xsd:string
```