The Vocabulary Mapping Framework (VMF): an introduction

v1.0, December 12, 2009

This document provides an introduction to the structure and development of the Vocabulary Mapping Framework (VMF) up to the end of the first stage of this work in November 2009. The document is in two parts: an **overview** and a **technical description.** Further details on the background to the JISC-funded project can be found on the project website at http://cdlr.strath.ac.uk/VMF/index.htm.

Table of contents

1. Overview

1.1 Purpose of the VMF

The initial aim of VMF is to provide a freely available tool which can be used to automatically compute the "best fit" mappings between terms in controlled vocabularies in different metadata schemes and messages (both standard and, in principle, proprietary) which are of interest to the educational, bibliographic and content publishing sectors. This tool is known as the **VMF matrix.** The ontology is likely to have other uses but this is the start point where there appears to be immediate practical benefit.

1.2 Scope of first release

The first release of the VMF matrix (the "alpha" release, as it is usable for experimentation but requires thorough practical testing, error-fixing and refinement) includes selected controlled vocabularies and parts of vocabularies from CIDOC CRM, DCMI, DDEX, FRAD, FRBR, IDF, LOM (IEEE), MARC21, MPEG21 RDD, ONIX and RDA as well as the complete RDA-ONIX Framework from which VMF is in part derived. URLs for the above can be found at the project website. The scope of VMF is not limited to these schemes and standards, but these are the initial focus, and many of them have representatives in the VMF project.

The initial scope of the mapped vocabularies is:

Resource categories (eg CD, Ebook, Photograph)
Resource-to-Resource relators (eg IsVersionOf, HasTranslation)
Resource-to-Party relators (eg Author, EditedBy)
Party-to-Party relators (eg AffiliatedTo)
Party categories

However, there is no constraint in principle on the VMF matrix being used to map vocabularies of any type. There is also no limitation in principle in the domains or vocabularies which might be mapped through the matrix, as the underlying ontology is generic.

Statistics for the v1.000 release of VMF:

- 928 Concept families including:
  - 1509 Resource Role concepts
  - 890 Party Role concepts
  - 11507 Relator concepts
- 824 terms mapped from third party vocabularies

Testing and updating of the matrix will be ongoing and changes will be incorporated in new numbered versions as needed. As an ontology, the VMF matrix should be viewed as data rather than software and so subject to routine updating.

The approach in this first stage has been "proof of concept", so groups of terms with quite diverse semantics from a variety of different schemes have been added to the matrix to test the methodology, rather than concentrating on very homogenous vocabularies which would give more complete but narrow results. The most similar vocabularies that have been mapped are those for Resource-to-Party contributor relations, and some exemplary one-to-one mapping results are shown in the appendix for Marc21 and ONIX vocabularies.

Attention has also been given to a thorough mapping of the classes in the CIDOC Content Reference Model (CRM), as this is the most comprehensive structured data model among the mapped schemas, and so provides a good test of the VMF method and the semantics of its basic terms. This mapping appears[1] to have been successful and has not raised any significant issues (though as with all mappings, some consultation with those responsible for the scheme will be needed to clarify some points).

[1] Proof of the effectiveness of any mappings will come only with more thorough testing.

Particular attention has also been given to mapping relators from the recently published RDA vocabularies. This is in part because they represent "state of the art" bibliographic metadata values, expressed in relators, but also because they are based on the FRBR data model. FRBR is challenging because it has three levels of abstraction for Resources (Work – Expression – Manifestation) rather than the two (Work – Manifestation) normally used in content industry schemes like ONIX or DDEX. Again the mappings appear to have been successful: the FRBR/RDA model is more granular but the matrix can support mapping between these two views.

1.3 Form of release

The VMF matrix as available for download from the VMF website as a single ontology in RDF triples (in the TTL format) using RDF, RDFS and OWL axioms, which may be viewed and edited in various ontology editing tools. Some comments on the use of the open source Protégé ontology editor are made in section 2.18.

There are two versions available on the VMF website:

VMF matrix complete (followed by version number and date)
VMF matrix without mappings (followed by version number and date)

The second contains the VMF ontology on its own without the mapped terms.

1.4 Structure of the VMF matrix

The matrix is a hierarchical class ontology of concepts grouped methodically using an event-based data model. This ontology can be extended as needed to provide a mapping point for any term in a vocabulary.

Terms from vocabularies are mapped into the matrix, not mapped directly to one another. Once a term is mapped onto the matrix, the internal links of the matrix establish computable relationships with every other mapped term in the matrix. The matrix therefore represents the sum total of all mapped concepts, plus other semantic relationships between them.

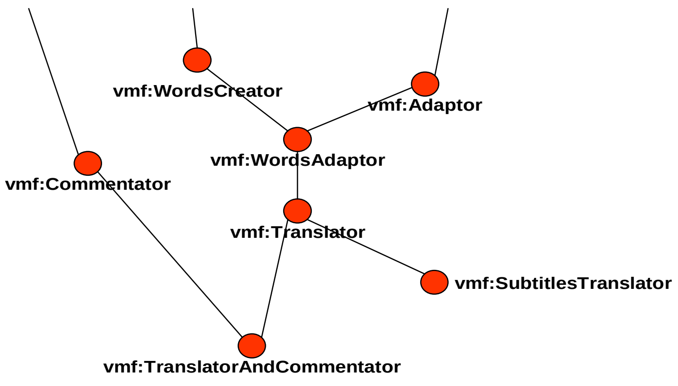
The matrix can then be queried, using SPARQL or another suitable language, to find the "best fit" direct mappings from one vocabulary to another. The current matrix does not include these direct "mappings out" from one vocabulary to another, but some illustrative results are provided (see Appendix).

The next stage of VMF is to refine the most useful kinds of queries, leading (in all likelihood) to the publication of recommended mappings for specific scheme pairs.

The VMF process is illustrated in simplified form in these three figures (note that the use of terms names in these figures are simplified for human readability):

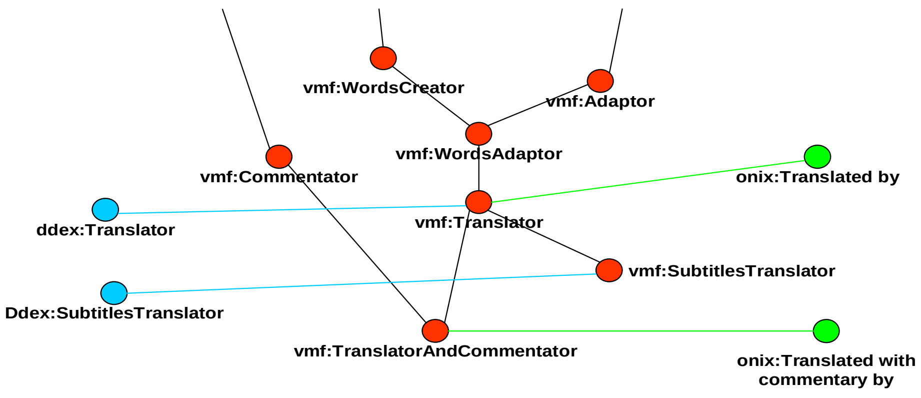
1. Creating the matrix

All concepts required to support mapping are added to the underlying VMF ontology using a rich contextual data model

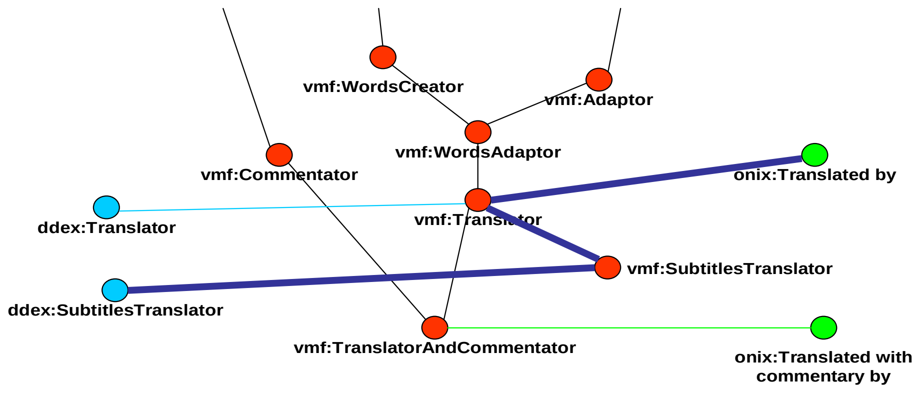


2. Mapping to VMF

Every term in a mapped vocabulary has a corresponding term in the VMF ontology



3. Mapping scheme to scheme
The matrix can be queried for the "best fit" starting from any point…



1.5 Authorization of mappings

In the initial release, the mappings in the VMF matrix are not authorized by any third parties. An essential part of the ongoing maintenance of VMF, if this is to happen, is that the participating schemes authorise the mappings of their own vocabularies to the VMF. Authorization means acknowledgement of the accuracy of the mappings. This process serves two key functions: first, it provides validation for, and correction of, the mappings themselves, and secondly it provides confidence for all participants and users. There is, of course, no such thing as objective accuracy in mapping where human understanding is involved, and so authorization represents "best endeavours" in this task.

As the VMF matrix will be freely available, there is no barrier to anyone attempting mappings or queries of their own for any purpose, and we encourage this to help in the development of the tool. However, it will not be sensible to allow mappings to be made in an ad hoc and unvalidated way ***if those mappings are going to be authoritative and used by others***. A mapping represents a statement of equivalence between the concepts of two different parties or domains, and both parties, or representatives of the domains, should give their assent to them if at all possible.

For this reason it is intended that a "canonical" copy of the matrix is maintained to which all authorised mappings are added.

Of course there is nothing to prevent a party mapping their own vocabularies into their own copy of the matrix for private use: but because the matrix will be changing it will be sensible for private mappings to be registered with the canonical version to ensure both authority and currency.

1.6 Ongoing maintenance and development

An approach for ongoing maintenance and development of the VMF is being developed by the Advisory Board. This includes:

• An organization willing to host and maintain the **VMF website** (the International DOI Foundation has expressed interest in this role).

• A **governance group**: the current project Advisory Board is a prototype for this.

• A **technical advisory group:** including technical representatives from the main participating schemes.

• A structure for enabling the **ongoing maintenance** and development of the matrix: this need not be costly, and costs should probably be borne by schemes or organizations wishing to map their vocabularies into the matrix, or to have them updated.

1.7 What the matrix is not

The matrix is a tool for computer, not human, use. It is a mapping tool, not a cataloguing tool or a public vocabulary. It is a very large network of terms whose job is to provide paths by which other terms may be connected: it is therefore not necessary for it to be generally accessible or "user-friendly" to users of metadata in general.

It is also **not** a dictionary of the public meanings of words, or an attempt to provide definitive meanings for particular words. In the VMF matrix each term has one precise meaning, and so each word can be a label for only one VMF concept, whereas in the world at large the same name may be associated with a range of diverse or related meanings, as is reflected in the various controlled vocabularies being mapped to VMF. Names are invaluable clues to the meaning of a term, but the unique meaning of a term is built up, and therefore recognised, by its definition and the accumulation of logical relationships in the ontology. Because VMF must represent the sum of its parts, it also becomes necessary for term names in VMF (which have to be unique) to be more precise, and therefore less user-friendly, than in a smaller scheme.

2. Technical description

2.1 Structure of the VMF matrix

The VMF matrix is a 'hub-and-spoke' ontology expressed in RDF triples[2]. At present data is initially prepared in an Excel workbook, from where it is automatically converted into RDF triples in the TTL format, in which form it can be viewed and processed with freely available tools such as the Protégé ontology editor and the Pellet OWL-DL reasoner.

The logical relationships within the matrix are expressed in a subset of the available RDF, RDFS and OWL axioms:

Table 1: RDF, RDFS and OWL axioms in VMF

rdfs:subClassOf	for relationships between a class and its parent(s)
rdfs:subPropertyOf	for relationships between a relator and its parent(s)
owl:equivalentClass	for mapping a class to its equivalent class in VMF
owl:equivalentProperty	for mapping a relator to its equivalent relator in VMF
rdfs:domain	for defining the class of the domain or subject of a relator
rdfs:range	for defining the class of the range or object of a relator
owl:inverseOf	for relating reciprocal relators
rdf:type	for identifying membership of classes
owl:disjointWith	for relating classes with no common members
owl:complementOf	for relating disjoint classes that make up a parent concept
owl:intersectionOf	for relating classes that are made by combining two or more concepts
owl:unionOf	for relating classes whose members may be either one thing or another

Each term that requires mapping from another vocabulary has a corresponding equivalent term in the matrix. These two terms are mapped using one of two relators **owl:equivalentClass** or **owl:equivalentProperty** according to whether the term is a class or a relator. For example[3]:

marc21:Relationship_Librettist owl:equivalentClass vmf:Librettist
ddex:ResourceContributorRole_Designer owl:equivalentProperty vmf:DesignedCreation_Designer

The VMF terms are joined to one another by logical relations including the parent/child relations of **rdfs:subClassOf** and **rdfs:subPropertyOf** according to whether the term is a class or a relator. For example:

vmf:Librettist rdfs:subClassOf vmf:WriterOfWordsToGoWithMusic
vmf: CreationDesign_Designer rdfs:subPropertyOf vmf:Plan_Planner

and so on.

The VMF matrix therefore is the sum of all the concepts which are mapped into it, plus a large number of other intermediate concepts which are needed to create computable relationships between the mapped terms according to the VMF data model.

2.2 Namespaces and term identification

Each term in the matrix is identified by a URI[4]. Some schemes (including DC and RDA) publish URIs for each of their terms. These are stored in the matrix. URIs for VMF terms will be published in due course. In the alpha release of the matrix we have used the unregistered

[2] It may be automatically transformed to be represented in other computable forms by those with the tools and motivation.
[3] The triple representation is slightly simplified here fo rreadability
[4] http://en.wikipedia.org/wiki/Uniform_Resource_Identifier

domain www.vmfmatrix.org to support URIs for all terms. This or some other appropriate domain will be registered in the next stage.

When a term from a vocabulary is mapped to the matrix, it is identified with a unique ID within the matrix. For example, terms from DDEX have IDs in this form:

> www.vmfmatrix.data/ddex#CreationType_MusicalWork.

These URIs are used only for managing internal matrix relationships. Because each term is *unique in the context of its own vocabulary*, the same term name has a different URI in different vocabularies within the same scheme .

This internal matrix ID is linked to a corresponding published URI if there is one, and so mappings can be input or output using those URIs. This scheme URI for a term is shown in the matrix using the relator **vmf:HasURI**. For example:

> rda:Creator-WorkRelator_designer vmf:HasURI
> http://RDVocab.info/RDARelationshipsWEMI/evaluatedInExpression

2.3 Human-readable names and annotations

The human readable label, definition, comments (if any) and short code or ID (if any) for each term are shown in the matrix using these relators following the term ID:

Table 2: Annotation relators

vmf:HasDisplayLabel	for a human readable name by which a concept is publicly known in its scheme.
vmf:HasDefinition	for the human readable definition or description of a concept in its scheme.
vmf:HasComment	for a human readable comment on the concept which may expand or exemplify the definition.
vmf:HasCode	For a short code used to identify a concept in its scheme.

For example:

> vmf:Derivation vmf:HasDefinition "A Creation made, in whole or part, from one or more existing Works."

2.4 The VMF data model

A requirement of the "hub and spoke" mapping approach is that the data model of the "hub" must be semantically rich enough to represent the meanings of all terms in the mapped vocabularies, and easily extensible to add new concepts as required. Because of the volume and complexity of mappings, without a clear model the VMF is likely to become unintelligible and unmaintainable.

2.4.1 Model antecedents

The matrix structure uses a standard model of formal ontology suited for logical inference based predominantly on attribute inheritance[5].

The non-formal or 'intensional' semantic concepts in the VMF matrix are based on Rightscom's COA[6] metamodel, which in turn is a development of the <indecs> metadata framework[7], and shares many common assumptions with FRBR and the CIDOC CRM. The COA model is used to support the maintenance of the DDEX standard, and indecs/COA has been the underlying

[5] John Sowa's website provides a useful and readable introduction to ontology and an example of a matrix model, see http://www.jfsowa.com/ontology/index.htm

[6] Contextual Ontology Architecture. The relevant parts of COA on which the matrix is based are all made explicit in this document as the vmf "concept family" model.

[7] http://en.wikipedia.org/wiki/Indecs_Content_Model

model for MPEG21 RDD, IDF metadata and recently for the ONIX for Publication Licenses standard[8].

The COA model is appropriate because, in a task that is inherently very complex, it provides *relative* simplicity[9]. The semantic relations within the VMF matrix are (compared with other large ontologies) clarified by the use of the COA's contextual model and its resulting **concept families.**

2.4.2 Classes, Individuals and Relators

Vocabulary terms to be mapped in the matrix are classes, individuals or relators.

The majority of mapped terms are **classes** (or *categories* or *types*) by which entities are classified according to one or more of their attributes (for example, Audiovisual Work, Person, Screensaver, Translator, Concordance, Erratum, MusicalArrangement, Payee, Owner, JPEG) .

A relatively small number of vocabularies deal with **individual** entities rather than members of classes. Most common of these are subject vocabularies which may include individuals such as William Shakespeare, the Eiffel Tower, the planet Jupiter or the French Revolution. The VMF matrix does not include individuals at this stage but can be extended to do so.

The remaining terms are **relators** describing relationships between two different resources or parties (for example, "is version of", "is author of", "is affiliate of"), and it is this requirement which presents the most interesting challenge, and for which the methodology of the **concept family**, explained in section 2.5, below is particularly well suited.

2.4.3 Relationships in the matrix

In the conventional way, classes are represented in a hierarchical matrix in which attributes may be inherited from one or more parent classes to build up more complex classes in the process of *specialization*. For example, the class of "Work" may be specialized to "VisualWork" and "InteractiveWork", and these two may be combined into the more specialized "InteractiveVisualWork". This is represented in the matrix as:

```
vmf:InteractiveWork  rdfs:subClassOf  vmf:Work
vmf:VisualWork  rdfs:subClassOf vmf:Work
vmf:InteractiveVisualWork  rdfs:subClassOf  vmf:InteractiveWork
vmf:InteractiveVisualWork  rdfs:subClassOf  vmf:VisualWork
```

This matrix of specialized classes may be extended to any level of granularity or complexity to support the particular terms to be mapped. Other logical relations (see Table 1) may be introduced to provide more precise definition and validation of the matrix. Statements of this kind are typically known as the **axioms** of the ontology.

2.4.4 Defining relators

There has been a growing trend towards the use of relators in metadata schemes: evidence for this is seen in the fact that FRBR, RDA and the CIDOC CRM are primarily built on relationships, and in the increasing use of the relationship-based RDF.

In fact, relators have always been widespread in metadata, but until recently have often been disguised as categories or roles. For example, all of the substantial contributor role vocabularies in schemes such as ONIX, MARC and DDEX are actually relators describing the relationship between the resource being described and some contributing party. The main reason for this sometimes superficial confusion lies in **naming**. For example, the ONIX contributor list (code list 17), which uses a mixture of role terms such as "Actor" and relator terms such as "Abridged

[8] http://www.editeur.org/21/ONIX-PL/

[9] Following Einstein: "Things should be made as simple as possible, but not simpler".

by", demonstrates this. "Actor" in this vocabulary is in fact a relator meaning "is actor in" (in context it is not saying that John Smith is an actor by profession, only that he acted in this particular resource). As a further example of the "hidden" prevalence of relators, of the Dublin Core 15 terms, only six describe wholly-owned attributes (title, identifier, description, type, format and coverage) while nine are relators from the resource to other independent entities (creator, contributor, publisher, source, relation, rights, date[10], language and subject, although the rights element is normally used in practise as a description and does not point to a particular entity). Metadata statements are increasingly recognized as being bi-directional: Shakespeare is metadata about Hamlet, and Hamlet is metadata about Shakespeare, depending on the starting point of view of any particular scheme, and the role of the relator in characterising relationships between entities is of course fundamental.

2.4.5 Relators and Events

Relationships between Classes (and therefore the Relators which name them) exist as a conequence of the **events** that bring entities into association with one another. An Event contains one or more entities playing a particular role (for example, a *creator* and a *creation* in a *creating event*). The relationships between theses classes (such as *is creator of* or *has creator*) are described by Relators. There is clearly a family relationship between these terms *create, creation, creating event* and Relators such as *is creator of* and *has creator*, based on the concept embodied in the verb *create.* The formal expression of these relationships in a **concept family** is the basis of the VMF data model.

Events may sometimes be expressed in a single relationship – for example, the simplest creating event only requires one creator and one creation, and so the single relationship "is creator of" (as in "Shakespeare is creator of Hamlet") may convey the full meaning of an event required by a particular metadata scheme.

However, other events involve three or more entities. For example, a deriving event (such as adapting or translating a resource) must have at least one agent (a "deriver" in VMF), at least one resource from which the derivation is made (a "source"), and at least one output (a "derivation"). This one event therefore gives rise to at least three relationships, and if they are described in both directions, then to six relators:

1. (deriver) is deriver of (derivation)
2. (derivation) is derived by (deriver)
3. (deriver) is deriver from (source)
4. (source) is derived from by (deriver)
5. (source) is source of (derivation)
6. (derivation) is derivation of (source)

Each of these relators may occur in some metadata scheme or other (often with a different name) and may require mapping to VMF (relators 1, 5 and 6 are the most common from this particular example).

In addition, where there are multiple derivers, sources or derivations in a particular event (such as the compilation of a series of CDs from a variety of tracks), there are further relationships:

7. (deriver) has co-deriver (deriver)
8. (derivation) has co-derivation (derivation)
9. (source) has co-source (source)

The addition then of a single further type of entity to the event (say, a "deriving tool" such as a computer) results in an arithmetic increase in the number of possible relationships which might be defined in some metadata scheme. An event with three role-playing classes has 9 possible

[10] Dates and times are almost universally regarded as attributes, but of course a time (whether a point in time or a period) is an independent entity in relation to which many things happen and exist.

relators, with four role-playing classes, with five classes 25, and so on. Each relator has a direct relationship with two classes, but an indirect relationship with every other one in the event, as it is impossible to adequately represent the concept (say) of a derivation without including the concepts of both a deriver and a source.

In the above example, thirteen terms (Derive, Deriver, Source, Derivation and nine relators) are all related through the single **concept** of deriving ("to make a new Creation from an existing Creation"). At least six of these terms (and large numbers of their children) appear regularly in vocabularies of different metadata schemes, and any of the others might occur occasionally. The VMF method of mapping relates all of these terms to their core concept, making it simpler to create, maintain and use in comparison to a conventional "flat" mapping between schemas which lacks a simple underlying model. The structure of the concept family enables this. This approach can be applied to **states** as well as events (a *State* in the VMF is a static multi-entity relationship such as rights ownership or whole-part relationships). Events and states are collectively called **contexts**[11] in the VMF.

So a family of classes and relators can be defined around a single 'verb' concept, known in VMF as a Concept Family[12]. The VMF matrix is built around Concept Families of terms, each based on a single verb, to achieve a 'simple as possible' and extensible framework for the mapping of large numbers of complex and at times highly granular terms.

2.5 Structure of a Concept Family

The VMF ontology is then a matrix of Concept Families organized within a hierarchy. The detailed structure of a Concept Family is given in the table below. Concept Families contain a small number of specialized types of concept:

Table 3: Concept Family components

Concept Type	Logical type	no per family	Description
Verb or Context Type	Class	1	A context in which some activity happens or some state persists
Agent	Class	0-n	An Entity playing an active role in the context.
Patient	Class	0-n	An Entity playing a passive role in the context.
Agent_Agent	Relator	0-n	A relator between an Agent and another Agent (only occurs if there can be at least two Agents in the ContextType).
Agent_Patient	Relator	0-n	A relator between an Agent and a Patient (only occurs if there are at least one of each).
Patient_Agent	Relator	0-n	A relator between a Patient and an Agent (only occurs if there are at least one of each).
Patient_Patient	Relator	0-n	A relator between a Patient and another Patient (only occurs if there can be at least two Patients in the ContextType).

Example of the Concept Family for "Adapt":

Table 4: Concept Family example

Concept Type	no	Term name	Definition
Verb or ContextType	1	Adapt / AdaptingEvent	To Derive an Adaptation.
Agent	1-n	Adapter	A Deriver of an Adaptation.
Patient	1-n	Adaptation	**A Derivation made by changing an existing Creation.**

[11] A Context in COA is actually "an intersection of time and place", and types of events and states are defined by the behaviour of verbs within a context. At this point time and place are not major issues for VMF, and so the VMF uses a "cut-down" version of the Context Model, which can be expanded in future if the need arises.
[12] Also known in other projects as a "Context Family"

	1-n	SourceOfAdaptation	The Source of an Adaptation.
Agent_Agent	0-n	Adapter_Adapter	The relator from one Adaptor to another in an Adapt Context .
Agent_Patient	1-n	Adapter_Adaptation	The relator from an Adaptor to an Adaptation in an Adapt Context.
	1-n	Adapter_SourceOfAdaptation	The relator from an Adaptor to a SourceOf Adaptation in an Adapt Context.
Patient_Agent	1-n	Adaptation_Adapter	The relator from an Adaptation to an Adaptor in an Adapt Context.
	1-n	SourceOfAdaptation_Adapter	The relator from a SourceOfAdaptation to an Adaptor in an Adapt Context.
Patient_Patient	1-n	Adaptation_SourceOfAdaptation	The relator from an Adaptation to a SourceOf Adaptation in an Adapt Context.
	0-n	Adaptation_Adaptation	The relator from one Adaptation to another in an Adapt Context.
	0-n	SourceOfAdaptation_ SourceOfAdaptation	The relator from a SourceOfAdaptation to another in an Adapt Context.

Notes on Table 4
1. In definitions, terms with capital initial are terms already defined in the ontology, usually as parents of the terms being defined.
2. Most meaning is inherited from parents: in this example, the parent family is Derive, and the meanings of Deriver, Derivation, Source etc are inherited from there.
3. The specialized meaning of a family is typically described in the definition of only **one** term (in this case, *Adaptation* – highlighted in bold). The other definitions are generally formulaic and reference that term.

The families themselves are linked in hierarchies, so that each term automatically knows its parents and children. For example, as Adapt is a child of Derive, then the Adapt family must contain a "child" member for each member of the Derive family: Adaptor is a child of Deriver, Adaptation is a child of Derivation, and SourceOfAdaptation is a child of Source.

2.6 Building the matrix

This section describes the practical steps involved in adding to the VMF matrix. When a new concept is identified as being required to support a mapping, the appropriate new verb is identified and related to its parent. If necessary, two or more families may be added at the same time to create the necessary conceptual hierarchy.

For example, because the concept of "translation" exists in a vocabulary to be mapped, "Translate" has been added as a subclass of "AdaptWords", meaning to adapt words by putting them into a different language. Once the verb concept is created, the Agent and Resource roles (in this case named "Translator" and "Translation") are named and added manually. The names and definitions of these terms are created manually, but their semantics are already fully derived through the semantics of "Translate" and the matrix inheritance model, so after definition of the concept and positioning it in the matrix, the process is routine.

The remaining terms and relationships are then generated automatically, including the names of all the relators, and all of the standard formal ontological relationships including parent classes and relators. For a concept such as Translate, that will include more than 50 ontological relationships. In addition, some more specific ontological relationships (see Logical axioms in Table 1) may be added manually in certain cases that support them.

The VMF matrix is then ready for mapping to any form of the concept "Translate" in any vocabulary, or for specializing it to a new concept family such as "TranslateSubtitles" or "TranslateToFrench" as needed. Event concepts may be as granular as required by the vocabulary being mapped. As typical examples, the ONIX product form vocabulary requires the concepts "EncodeBetamaxVideocassetteInSECAM" and "FixRolledSheetMap", each of which inherits meaning from a range of more general concepts within the matrix (fixing, encoding, Betamax, Videocassette, maps, Sheet carriers, rolled sheets, SECAM). Some vocabularies require concepts of much greater granularity.

The methodology therefore allows for the rapid generation of large numbers of formally related terms which are "nodes" in the matrix, any of which may be mapped to a term in an external vocabulary as required, thereby bringing that term fully into the matrix.

2.7 Concept Families for attributes

Attributes are added to the matrix by being represented in a new class which is a member of a family. For example, each of the RDA ONIX categories is represented as a class, so that the attributes of being (say) "Interactive" and "NonInteractive" are represented as classes of "InteractiveCreation" and "NonInteractiveCreation", which then become parents of any other classes sharing those attributes.

2.8 Upper ontology

The "upper ontology" of the matrix (that is, the high level terms on which it is based) is adapted directly from verbs in Rightscom's COA ontology, representing commonplace concepts. Note that there is no formal distinction between the "upper ontology" and the rest of the matrix: the distinction is merely an arbitrary convenience: new primitive semantics are introduced and mapped as required at whatever level.

2.9 Specialization of concepts

A concept is specialized, and a new family therefore created, in a number of ways. The main ones are given in Table 5:

Table 5: methods of specialization

Primitive Semantics (see also 2.10)	A new concept is introduced into the ontology and combined with existing concepts. For example, the verb Make adds the concept of "bringing something into existence" to the concept of Do.
Intersection	Two or more existing concepts are combined to form a new one. For example, CreateWords and Adapt are combined into AdaptWords.
Union	A concept is defined as being one out of two more other concepts. For example TakeFilmOrPhotograph is the union of TakeFilm and TakePhotograph.
Disjunction	A concept is defined as being disjoint with another with a common parent (that is, one individual cannot belong to both classes). For example, OriginalWork is disjoint with DerivedWork.
Cardinality Constraint	A concept is specialized because of the number of occurrences of a role within it. For example, "Act" must have at least an Agent or a Patient, but not necessarily both. "Do" is a specialization of Act which must have at least one Agent.
Antecedent	A concept defined by the state which arises as a consequence of it.
Consequent	A concept defined by the event which causes it.
Conditional Rule (see also 2.13)	A concept defined with some other conditional rule, including measurement, temporality or modality. At this stage conditional rules are not explicit in the ontology but will be added in the next stage of development.
Dependent Role (see also 2.14)	A concept is specialized because one of its Resources has

The method of specialization of each concept is shown in the ontology using the relator **vmf:HasDifferentiae** following the term ID ("differentiae" being the classical term for the point of specialization). For example:

vmf:Derivation vmf:HasDifferentiae vmf:PrimitiveSemantics

2.10 Primitive semantics

The specialization methods used in VMF are common in ontologies and taxonomies. Behind them lies the principle that a new element of meaning (called **primitive semantics** in the VMF) should be introduced only once into the matrix, and then inherited (or otherwise logically connected with axioms) by every concept which includes or otherwise relies upon it. In the example above, the concept of being "visual" (that is, being perceivable by the sense of sight) is not introduced into the matrix with the term "VisualWork", because many things other than Works may be visual, so there is a simpler concept of a "SeeableResource" at a higher point in the matrix, from which "VisualWork" inherits this concept.

Primitive semantics may be very general (such as the concepts of sequential time, animate life, or of compiling a new creation from parts of existing ones) or they may be very specific (such as the a baby grand piano, the HTML markup language or a Pantone colour reference). Primitive semantics are concepts that must be agreed (explicitly or implicitly) by those who are agreeing to any mapping. For example, if onix:Librettist and marc21:Librettist are mapped as equivalent to vmf:Librettist in the VMF, it is because it is assumed that the inherited concepts that make up vmf:Librettist (which include opera, words, music and creating) are shared by those three schemes. Primitive semantics normally form part of the definition of a term, but in VMF they are also identified explicitly using the relator **vmf:HasPrimitiveSemantics**. For example:

vmf:Derivation vmf:HasPrimitiveSemantics "Derivation: A Creation can be made from a pre-existing Work."

The reason for this approach is that a user of VMF should be able to verify that they agree with the primitive semantics included without needing to understand the matrix that distributes them.

2.11 Concept Family axioms

Each Concept Family is itself identified as a concept (for example, "Adapt_CF"). VMF uses a small set of relators (shown in Table 6) to express the relationships between the different elements and their Concept Family.

Table 6: Concept Family relators

vmf:IsContextInCF	to relate a Verb to its Concept Family (for example vmf:Adapt vmf:IsContextInCF vmf:Adapt_CF)
vmf:IsResourceInCF	to relate a Resource Role (either an Agent or Patient) and its Concept Family (for example vmf:Adaptation vmf:IsResourceInCF vmf:Adapt_CF)
vmf:IsRelatorInCF	to relate a Relator and its Concept Family (for example vmf:Adaptation_Adaptor vmf:IsRelatorInCF vmf:Adapt_CF)

These relators enable relationships to be established between different roles in a Context.

2.12 Conditional Rules

At this point all axioms are expressed as class-to-class relationships: that is, there is no rulebase in which conditional rules containing rdf:Type statements are made about variables representing instances of a Class. This means that certain ontological relationships are not yet logically explicit in the matrix. These relationships are currently expressed either through the Concept Family axioms (2.11), or else as primitive semantics (2.10), or else informally in comments. For the purpose of mapping vocabulary terms into the matrix, the class hierarchy is adequate, but the intention is to introduce a rulebase in the next stage of the VMF to maximise the effectiveness of the production of scheme-to-scheme mappings.

2.13 Dependent Roles

This section describes how the matrix currently deals with an important semantic issue: inherited attributes of a class which fall outside of the immediate context, particularly **fixed type** attributes. A fixed type (sometimes known as a "natural type") is a static category to which something belongs for a period of its existence (often for the whole of its existence), in contrast to a **role** which it only plays in a particular context. For example, John Smith may have a fixed type of "Human" all his life, but plays a role of "Composer" only at those times when he is composing music.

Fixed types are handled somewhat unusually in VMF. Because all classes are defined in verb-based contexts, a fixed type is identified in the event in which it came into being, and it then persists in the subsequent state which follows the event. For example, a MusicalWork comes into being in a Compose event; once it is created, it remains permanently in the state of being a MusicalWork, but is no longer being created.

When defining a concept in the matrix it is necessary to know whether attributes inherited by the classes in the concept family are inherited from an event in the past, present or future. For example, if a new MusicalWork (say, an Arrangement) is derived from an existing one (its Source), then it is true that both Source and Arrangement are types of MusicalWork, but with a critical difference: they do not become Works in the same Event. In the Arrange event, only the Arrangement is coming into existence: the Source came into existence in an earlier Compose event. If both are simply defined as being a subclass of Work that would ignore the temporal distinctions and destroy any hope of accurate inference - for example, some types of query would infer that both the Arrangement and its Source were created by the Arranger, which would be incorrect.

There are a variety of ways in which these temporal constraints may be computed. At this stage the matrix simply records any "non-family" inheritance relationship with one or these specialized relators:

Table 7: Dependent Role relators

vmf:HasPastRole	A role in a Context which ended before the current one began.
vmf:HasConcurrentRole	A role in a Context which occurs throughout or within the current one.
vmf:HasFutureRole	A role in a Context which starts at a time following the end of current one.
vmf:HasPastOrConcurrentRole	A role in a past or concurrent context.
vmf:HasPastOrFutureRole	A role in a past or future context.
vmf:HasConcurrentOrFutureRole	A role in a concurrent or future context.
vmf:HasAnytimeRole	A role in a context which may occur at any time.

For example:

vmf:Source vmf:HasPastRole vmf:Work

This is used in contrast to:

vmf:Arrangement rdfs:subClassOf vmf:Work

These relators may be transformed into explicit conditional rules in future processing of the matrix.

2.14 Displaced relationships

Another important type of conditional rule that is required is to deal with **displaced relationships.** These are relationships which create "short cuts" between two Entities which are more accurately related through a chain of two or more relationships. For example, many metadata schemes have relators which associate a creator of a Work with the creation of a later Adaptation or Manifestation of that work: for example, the writer of a novel may be linked as the "author of original work" to a later translation, or the composer of a piece of music may be linked directly to a recording of a performance. These relators are expressed in the matrix (this particular example appears in the matrix as the concept "CreateSource") but require rules which allow them to be transformed into their fuller expression.

2.15 Membership of vocabularies

The relationship between a term and the vocabulary to which it belongs is described with the relator **vmf:IsInVocabulary.** For example

onix:CodeList17_By_author vmf:IsInVocabulary onixAVS:CodeList17.

2.16 Concept names

Human-readable names for concepts in VMF follow conventions and, in the case of relators, automated rules. This makes it easier to create and find them.

Contexts are named with the infinitive form of an English language verb (for example *Create* or *CompileWords*). Agent and Resource roles are nouns (for example, *Creator, Creation*).

Relators are named by combining the names of the two related concepts, separated by an underscore (for example, *Creator_Creation, CompiledWords_WordsCompiler*). This results in some apparent redundancy in the name, as the same semantic concept is implied on both sides of the relator, but because the names are not intended for public use this is not an issue.

Concepts in the same family almost invariably retain the same linguistic 'stem' (for example, *Create, Creator, Creation*).

2.17 QA and validation

The matrix may be validated for logical consistency using OWL-DL reasoners. Mapping is the means by which the matrix is extended and corrected, and mapping a new term forces the scrutiny and validation of the existing matrix. Mapping may result in the addition of new "intermediate" or "leaf" concepts (a leaf concept is one with no children). Because the most useful terms are most often scrutinized, this is an effective method of QA. Ultimately the accuracy of the matrix will be tested by its results in terms of scheme-to-scheme mappings.

2.18 Producing the matrix output

The RDF database is generated at present by applying Excel macros to the VMF spreadsheet containing and producing triples in RDF TTL ("Turtle") syntax. An additional process applying a SPARQL construct rule is required for completing the rdfs:subPropertyOf hierarchies.

In the compilation process the open source Protégé editor was used for reviewing the ontology. Note that the standard visualization plug-in has specific limitations for VMF: it enables the viewer to review normal subclass hierarchies and mappings but not the concept family groupings which are key to the matrix.

An example of the RDF triples currently generated for the Concept Family "vmf:Write_CF", including mappings of several terms from different schemes, is shown below. A tabular version of the same Concept Family is shown in Table 8 following.

```
# classes
vmf:CreateLexicalWork rdfs:subClassOf vmf:CreateWork.
vmf:CreateLexicalWork vmf:HasReferenceName "CreateLexicalWork".
vmf:CreateLexicalWork vmf:HasSynonym "WordsCreatingEvent".
vmf:CreateLexicalWork vmf:IsContextInCF vmf:CreateLexicalWork_CF.
vmf:CreateLexicalWork_CF vmf:HasDefinition "The ConceptFamily of CreateLexicalWork".
vmf:CreateLexicalWork_CF vmf:HasReferenceName "CreateLexicalWork_CF".
vmf:CreatorOfLexicalWork rdfs:subClassOf vmf:CreatorOfWork.
vmf:CreatorOfLexicalWork vmf:HasReferenceName "CreatorOfLexicalWork".
vmf:CreatorOfLexicalWork vmf:IsAgentInCF vmf:CreateLexicalWork_CF.
vmf:LexicalWork rdfs:subClassOf vmf:Work.
vmf:LexicalWork vmf:HasReferenceName "LexicalWork".
```

```
vmf:LexicalWork vmf:IsResourceInCF vmf:CreateLexicalWork_CF.

# relators
vmf:CreateLexicalWork_CreatorOfLexicalWork rdfs:domain vmf:CreateLexicalWork.
vmf:CreateLexicalWork_CreatorOfLexicalWork rdfs:range vmf:CreatorOfLexicalWork.
vmf:CreateLexicalWork_CreatorOfLexicalWork rdfs:subPropertyOf vmf:CreateWork_CreatorOfWork.
vmf:CreateLexicalWork_CreatorOfLexicalWork vmf:HasReferenceName
"CreateLexicalWork_CreatorOfLexicalWork".
vmf:CreateLexicalWork_CreatorOfLexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:CreateLexicalWork_LexicalWork rdfs:domain vmf:CreateLexicalWork.
vmf:CreateLexicalWork_LexicalWork rdfs:range vmf:LexicalWork.
vmf:CreateLexicalWork_LexicalWork rdfs:subPropertyOf vmf:CreateWork_Work.
vmf:CreateLexicalWork_LexicalWork vmf:HasReferenceName "CreateLexicalWork_LexicalWork".
vmf:CreateLexicalWork_LexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:CreatorOfLexicalWork_CreateLexicalWork owl:inverseOf vmf:CreateLexicalWork_CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_CreateLexicalWork rdfs:domain vmf:CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_CreateLexicalWork rdfs:range vmf:CreateLexicalWork.
vmf:CreatorOfLexicalWork_CreateLexicalWork rdfs:subPropertyOf vmf:CreatorOfWork_CreateWork.
vmf:CreatorOfLexicalWork_CreateLexicalWork vmf:HasReferenceName
"CreatorOfLexicalWork_CreateLexicalWork".
vmf:CreatorOfLexicalWork_CreateLexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork owl:inverseOf vmf:CreatorOfLexicalWork_CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork rdfs:domain vmf:CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork rdfs:range vmf:CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork rdfs:subPropertyOf vmf:CreatorOfWork_CreatorOfWork.
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork vmf:HasReferenceName
"CreatorOfLexicalWork_CreatorOfLexicalWork".
vmf:CreatorOfLexicalWork_CreatorOfLexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:CreatorOfLexicalWork_LexicalWork owl:inverseOf vmf:LexicalWork_CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_LexicalWork rdfs:domain vmf:CreatorOfLexicalWork.
vmf:CreatorOfLexicalWork_LexicalWork rdfs:range vmf:LexicalWork.
vmf:CreatorOfLexicalWork_LexicalWork rdfs:subPropertyOf vmf:CreatorOfWork_Work.
vmf:CreatorOfLexicalWork_LexicalWork vmf:HasReferenceName "CreatorOfLexicalWork_LexicalWork".
vmf:CreatorOfLexicalWork_LexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:LexicalWork_CreateLexicalWork owl:inverseOf vmf:CreateLexicalWork_LexicalWork.
vmf:LexicalWork_CreateLexicalWork rdfs:domain vmf:LexicalWork.
vmf:LexicalWork_CreateLexicalWork rdfs:range vmf:CreateLexicalWork.
vmf:LexicalWork_CreateLexicalWork rdfs:subPropertyOf vmf:Work_CreateWork.
vmf:LexicalWork_CreateLexicalWork vmf:HasReferenceName "LexicalWork_CreateLexicalWork".
vmf:LexicalWork_CreateLexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:LexicalWork_CreatorOfLexicalWork rdfs:domain vmf:LexicalWork.
vmf:LexicalWork_CreatorOfLexicalWork rdfs:range vmf:CreatorOfLexicalWork.
vmf:LexicalWork_CreatorOfLexicalWork rdfs:subPropertyOf vmf:Work_CreatorOfWork.
vmf:LexicalWork_CreatorOfLexicalWork vmf:HasReferenceName "LexicalWork_CreatorOfLexicalWork".
vmf:LexicalWork_CreatorOfLexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.
vmf:LexicalWork_LexicalWork owl:inverseOf vmf:LexicalWork_LexicalWork.
vmf:LexicalWork_LexicalWork rdfs:domain vmf:LexicalWork.
vmf:LexicalWork_LexicalWork rdfs:range vmf:LexicalWork.
vmf:LexicalWork_LexicalWork rdfs:subPropertyOf vmf:Work_Work.
vmf:LexicalWork_LexicalWork vmf:HasReferenceName "LexicalWork_LexicalWork".
vmf:LexicalWork_LexicalWork vmf:IsRelatorInCF vmf:CreateLexicalWork_CF.

# definitions, primitive semantics and differentiae
vmf:CreateLexicalWork vmf:HasDefinition "To Create a LexicalWork.".
vmf:CreateLexicalWork vmf:HasDifferentiae "Rule”.
vmf:CreatorOfLexicalWork vmf:HasDefinition "A Creator of a LexicalWork.".
vmf:LexicalWork vmf:HasDefinition "A Work which may be realized in language.".

# mappings
ddex:MusicalWorkContributorRole_Author owl:equivalentProperty vmf:LexicalWork_CreatorOfLexicalWork.
ddex:MusicalWorkContributorRole_Author rdfs:subPropertyOf vmf:DdexTerm.
ddex:MusicalWorkContributorRole_Author vmf:HasDescription "A Creator of written or spoken words which form
part of a Resource.".
ddex:MusicalWorkContributorRole_Author vmf:IsInVocabulary ddexC:MusicalWorkContributorRole.
```

onix:CodeList17_By__author_ owl:equivalentProperty vmf:LexicalWork_CreatorOfLexicalWork.
onix:CodeList17_By__author_ rdfs:subPropertyOf vmf:OnixTerm.
onix:CodeList17_By__author_ vmf:HasAnnotation "Author of a textual work".
onix:CodeList17_By__author_ vmf:HasCode "A01".
onix:CodeList17_By__author_ vmf:HasDescription "By (author)".
onix:CodeList17_By__author_ vmf:IsInVocabulary onixAVS:CodeList17.
rda:Creator-WorkRelator_author owl:equivalentProperty vmf:LexicalWork_CreatorOfLexicalWork.
rda:Creator-WorkRelator_author rdfs:subPropertyOf vmf:RdaTerm.
rda:Creator-WorkRelator_author vmf:HasDescription "A person, family, or corporate body responsible for creating a work that is primarily textual in content, regardless of media type (e.g., printed text, spoken word, electronic text, tactile text) or genre (e.g., poems, novels, screenplays, blogs). Use also for persons, etc., creating a new work by paraphrasing, rewriting, or adapting works by another creator such that the modification has substantially changed the nature and content of the original or changed the medium of expression.".
rda:Creator-WorkRelator_author vmf:IsInVocabulary rdaAVS:Creator-WorkRelator.
rdaonix:Character_language owl:equivalentClass vmf:LexicalWork.
rdaonix:Character_language rdfs:subClassOf vmf:RdaonixTerm.
rdaonix:Character_language vmf:HasDescription "Content expressed in human or machine-readable language.".
rdaonix:Character_language vmf:IsInVocabulary rdaonixAVS:Character.

Tabular representation of the above.

Table 8: Tabular representation of Concept Family example

Concept Family for CreateLexicalWork			
Verb	**CreateLexicalWork**	synonym	WordsCreatingEvent
		parent	CreateWork
		definition	To Create a LexicalWork.
		differentiae	1. Primitive semantics
		primitive semantics	1. Language: concepts may be expressed in words.
Agent Role	**CreatorOfLexicalWork**	parent	CreatorOfWork
		definition	A Creator of the lexical elements of a LexicalWork.
Resource Role	**LexicalWork**	parent	Work
		definition	A Work expressed in language.
Relators	**CreateLexicalWork_CreatorOfLexicalWork**	domain	CreateLexicalWork
		range	CreatorOfLexicalWork
		reciprocal	CreatorOfLexicalWork_CreateLexicalWork
		parent	CreateWork_CreatorOfWork
	CreateLexicalWork_LexicalWork	domain	CreateLexicalWork
		range	LexicalWork
		reciprocal	LexicalWork_CreateLexicalWork
		parent	CreateWork_Work
	CreatorOfLexicalWork_CreatorOfLexicalWork	domain	CreatorOfLexicalWork
		range	CreatorOfLexicalWork
		reciprocal	CreatorOfLexicalWork_CreatorOfLexicalWork
		parent	CreatorOfWork_CreatorOfWork
	CreatorOfLexicalWork_LexicalWork	domain	CreatorOfLexicalWork
		range	LexicalWork
		reciprocal	LexicalWork_CreatorOfLexicalWork
		parent	CreatorOfWork_Work
	CreatorOfLexicalWork_CreateLexicalWork	domain	CreatorOfLexicalWork
		range	CreateLexicalWork
		reciprocal	CreateLexicalWork_CreatorOfLexicalWork

		parent	CreatorOfWork_CreateWork
	LexicalWork_CreatorOfLexicalWork	domain	LexicalWork
		range	CreatorOfLexicalWork
		reciprocal	CreatorOfLexicalWork_LexicalWork
		parent	Work_CreatorOfWork
	LexicalWork_LexicalWork	domain	LexicalWork
		range	LexicalWork
		reciprocal	LexicalWork_LexicalWork
		parent	Work_Work
	LexicalWork_CreateLexicalWork	domain	LexicalWork
		range	CreateLexicalWork
		reciprocal	CreateLexicalWork_LexicalWork
		parent	Work_Create
Mappings	**rdaonix:Character_language**	same as	LexicalWork
		definition	Content expressed in human or machine-readable language.
		vocabulary	rdaonixAVS:Character
	rda:Creator-WorkRelators_author	same as	LexicalWork_CreatorOfLexicalWork
		display label	author
		definition	A person, family, or corporate body responsible for creating a work that is primarily textual in content, regardless of media type (e.g., printed text, spoken word, electronic text, tactile text) or genre (e.g., poems, novels, screenplays, blogs). Use also for persons, etc., creating a new work by paraphrasing, rewriting, or adapting works by another creator such that the modification has substantially changed the nature and content of the original or changed the medium of expression.
		vocabulary	rdaAVS:Creator-WorkRelators
	ddex:MusicalWorkContributorRole_Author	same as	LexicalWork_CreatorOfLexicalWork
		display label	Author
		definition	A Creator of written or spoken words which form part of a Resource.
		vocabulary	ddexC:MusicalWorkContributorRole.
	onix:CodeList17_By_author	same as	LexicalWork_CreatorOfLexicalWork
		display label	By (author)
		code	A01
		definition	Author of a textual work.
		vocabulary	onixAVS:CodeList17

3. Producing scheme to scheme mappings

This is the proposed task of the next stage of the project. As a first step, some SPARQL queries have been created and an illustrative mapping is provided in section 3.1.

3.1 Example mappings

This section shows candidate mappings between Onix Code List 17 and the Marc 21 "Relationship" vocabularies. The results are partial as not all terms from these two large vocabularies have yet been mapped to the VMF matrix. The SPARQL queries used to derive the mappings are shown following.

Exact equivalence mappings are shown in bold and "best fit" options in medium font.

Marc 21 Relationship		**Onix Code List 17**
Actor	**same as**	**Actor**
Actor	parent	Performed by
Adapter	child	Dramatized by
Adapter	parent	Adapted by
Adapter	sibling	Abridged by
Adapter	sibling	Other adaptation by
Adapter	sibling	Translated by
Architect	parent	Designed by
Architect	sibling	Cover design or artwork by
Arranger	**same as**	**Arranged by music**
Arranger	parent	By composer
Arranger	sibling	Adapted by
Artist	**same as**	**By artist**
Artist	sibling	Drawings by
Author	child	By author
Author	child	By composer
Author	child	Software written by
Author	sibling	From an idea by
Author of introduction etc	child	Introduction by
Author of introduction etc	sibling	Commentaries by
Author of introduction etc	sibling	Memoir by
Author of introduction etc	sibling	Notes by
Author of introduction etc	sibling	Summary by
Author of screenplay	**same as**	**Screenplay by**
Author of screenplay	sibling	Dramatized by
Cartographer	**same as**	**Maps by**
Cartographer	parent	By author
Cartographer	sibling	Adapted by
Cartographer	sibling	Original author
Cartographer	sibling	Text by
Commentator	**same as**	**Commentator**
Commentator for written text	sibling	Commentaries by

Compiler	**same as**	**Compiled by**
Compiler	child	Other compilation by
Composer	**same as**	**By composer**
Composer	child	Arranged by music
Composer	sibling	By author
Composer	sibling	Software written by
Conceptor	**same as**	**From an idea by**
Conductor	**same as**	**Conductor**
Contributor	**same as**	**Contributions by**
Contributor	child	Producer
Contributor	parent	Created by
Contributor	sibling	Director
Contributor	sibling	Edited by
Costume designer	parent	Designed by
Costume designer	sibling	Cover design or artwork by
Cover designer	parent	Designed by
Cover designer	sibling	Cover design or artwork by
Creator	**same as**	**Created by**
Creator	child	Contributions by
Creator	child	Director
Creator	child	Edited by
Dancer	**same as**	**Dancer**
Designer	**same as**	**Designed by**
Designer	child	Cover design or artwork by
Director	**same as**	**Director**
Director	child	Other direction by
Director	parent	Created by
Director	sibling	Contributions by
Director	sibling	Edited by
Draftsman	child	Maps by
Draftsman	sibling	Designed by
Editor	**same as**	**Edited by**
Editor	parent	Created by
Editor	sibling	Contributions by
Editor	sibling	Director
Illustrator	**same as**	**Illustrated by**
Illustrator	parent	Drawings by
Illustrator	sibling	Text by
Instrumentalist	child	Instrumental soloist
Instrumentalist	sibling	Performed by orchestra band ensemble
Landscape architect	parent	Designed by
Landscape architect	sibling	Cover design or artwork by

Librettist	**same as**	**Libretto by**
Librettist	sibling	Lyrics by
Lyricist	**same as**	**Lyrics by**
Lyricist	child	Book and lyrics by
Lyricist	sibling	Libretto by
Musical director	child	Conductor
Musical director	sibling	Conductor
Musician	child	Performed by orchestra band ensemble
Narrator	**same as**	**Narrator**
Narrator	sibling	Read by
Originator	sibling	By author
Originator	sibling	By composer
Originator	sibling	Software written by
Performer	**same as**	**Performed by**
Performer	child	Actor
Photographer	child	Filmed photographed by
Programmer	parent	Software written by
Publisher	child	Performed by
Reporter	child	General rapporteur
Reporter	parent	By author
Reporter	sibling	Adapted by
Reporter	sibling	Maps by
Reporter	sibling	Original author
Reporter	sibling	Text by
Singer	child	Vocal soloist
Singer	sibling	Performed by orchestra band ensemble
Speaker	child	Narrator
Speaker	child	Read by
Storyteller	sibling	Narrator
Storyteller	sibling	Read by
Transcriber	sibling	Compiled by
Translator	**same as**	**Translated by**
Translator	child	Edited and translated by
Translator	parent	Adapted by
Translator	sibling	Abridged by
Translator	sibling	Other adaptation by

Onix Code List 17		**Marc 21 Relationship**
Abridged by	sibling	Adapter
Abridged by	sibling	Translator
Actor	**same as**	**Actor**
Actor	parent	Performer
Adapted by	child	Adapter
Adapted by	child	Translator
Adapted by	sibling	Arranger
Adapted by	sibling	Cartographer
Adapted by	sibling	Reporter
Arranged by music	**same as**	**Arranger**
Arranged by music	parent	Composer
Book and lyrics by	parent	Lyricist
By artist	**same as**	**Artist**
By author	child	Cartographer
By author	child	Reporter
By author	parent	Author
By author	sibling	Composer
By author	sibling	Originator
By composer	same as	Composer
By composer	child	Arranger
By composer	parent	Author
By composer	sibling	Originator
Commentaries by	sibling	Author of introduction etc
Commentaries by	sibling	Commentator for written text
Commentator	**same as**	**Commentator**
Compiled by	**same as**	**Compiler**
Compiled by	sibling	Transcriber
Conductor	**same as**	**Conductor**
Conductor	parent	Musical director
Conductor	sibling	Musical director
Contributions by	**same as**	**Contributor**
Contributions by	parent	Creator
Contributions by	sibling	Director
Contributions by	sibling	Editor
Cover design or artwork by	parent	Designer
Cover design or artwork by	sibling	Architect
Cover design or artwork by	sibling	Costume designer
Cover design or artwork by	sibling	Cover designer
Cover design or artwork by	sibling	Landscape architect

Created by	**same as**	**Creator**
Created by	child	Contributor
Created by	child	Director
Created by	child	Editor
Dancer	**same as**	**Dancer**
Designed by	child	Architect
Designed by	child	Costume designer
Designed by	child	Cover designer
Designed by	child	Landscape architect
Designed by	same as	Designer
Designed by	sibling	Draftsman
Director	**same as**	**Director**
Director	parent	Creator
Director	sibling	Contributor
Director	sibling	Editor
Dramatized by	parent	Adapter
Dramatized by	sibling	Author of screenplay
Drawings by	child	Illustrator
Drawings by	sibling	Artist
Edited and translated by	parent	Translator
Edited by	**same as**	**Editor**
Edited by	parent	Creator
Edited by	sibling	Contributor
Edited by	sibling	Director
Filmed photographed by	parent	Photographer
From an idea by	**same as**	**Conceptor**
From an idea by	sibling	Author
General rapporteur	parent	Reporter
Illustrated by	**same as**	**Illustrator**
Instrumental soloist	parent	Instrumentalist
Introduction by	parent	Author of introduction etc
Libretto by	**same as**	**Librettist**
Libretto by	sibling	Lyricist
Lyrics by	**same as**	**Lyricist**
Lyrics by	sibling	Librettist
Maps by	**same as**	**Cartographer**
Maps by	parent	Draftsman
Maps by	sibling	Reporter
Memoir by	sibling	Author of introduction etc

Narrator	**same as**	**Narrator**
Narrator	parent	Speaker
Narrator	sibling	Storyteller
Notes by	sibling	Author of introduction etc
Original author	sibling	Cartographer
Original author	sibling	Reporter
Other adaptation by	sibling	Adapter
Other adaptation by	sibling	Translator
Other compilation by	parent	Compiler
Other direction by	parent	Director
Performed by	**same as**	**Performer**
Performed by	child	Actor
Performed by	parent	Publisher
Performed by orchestra band ensemble	parent	Musician
Performed by orchestra band ensemble	sibling	Instrumentalist
Performed by orchestra band ensemble	sibling	Singer
Producer	parent	Contributor
Read by	parent	Speaker
Read by	sibling	Narrator
Read by	sibling	Storyteller
Screenplay by	**same as**	**Author of screenplay**
Software written by	child	Programmer
Software written by	parent	Author
Software written by	sibling	Composer
Software written by	sibling	Originator
Summary by	sibling	Author of introduction etc
Text by	sibling	Cartographer
Text by	sibling	Illustrator
Text by	sibling	Reporter
Translated by	**same as**	**Translator**
Translated by	sibling	Adapter
Vocal soloist	parent	Singer

SPARQL Queries used to create candidate mappings

```
select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
{?InTerm owl:equivalentClass ?VmfTerm}
   UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
?InTerm vmf:IsInVocabulary ?InAVS .
{?OutTerm owl:equivalentClass ?VmfTerm}
   UNION {?OutTerm owl:equivalentProperty ?VmfTerm} .
?OutTerm vmf:IsInVocabulary  ?OutAVS .
filter(?InAVS=<marc21AVS#Relationship>) .
filter(?OutAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
{?InTerm owl:equivalentClass ?VmfTerm}
   UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
?InTerm vmf:IsInVocabulary ?InAVS .
{?OutTerm owl:equivalentClass ?VmfTerm}
   UNION {?OutTerm owl:equivalentProperty ?VmfTerm} .
?OutTerm vmf:IsInVocabulary  ?OutAVS .
filter(?OutAVS=<marc21AVS#Relationship>) .
filter(?InAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfTerm rdfs:subClassOf ?VmfParent.
   ?VmfSib rdfs:subClassOf ?VmfParent}
     UNION {?VmfTerm rdfs:subPropertyOf ?VmfParent.
     ?VmfSib rdfs:subPropertyOf ?VmfParent} .
  {?OutTerm owl:equivalentClass ?VmfSib}
     UNION {?OutTerm owl:equivalentProperty ?VmfSib} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?InAVS=<marc21AVS#Relationship>) .
filter(?OutAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfTerm rdfs:subClassOf ?VmfParent.
   ?VmfSib rdfs:subClassOf ?VmfParent}
     UNION {?VmfTerm rdfs:subPropertyOf ?VmfParent.
     ?VmfSib rdfs:subPropertyOf ?VmfParent} .
  {?OutTerm owl:equivalentClass ?VmfSib}
     UNION {?OutTerm owl:equivalentProperty ?VmfSib} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?OutAVS=<marc21AVS#Relationship>) .
filter(?InAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
```

```
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfChild rdfs:subClassOf ?VmfTerm}
     UNION {?VmfChild rdfs:subPropertyOf ?VmfTerm} .
  {?OutTerm owl:equivalentClass ?VmfChild}
     UNION {?OutTerm owl:equivalentProperty ?VmfChild} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?InAVS=<marc21AVS#Relationship>) .
filter(?OutAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfChild rdfs:subClassOf ?VmfTerm}
     UNION {?VmfChild rdfs:subPropertyOf ?VmfTerm} .
  {?OutTerm owl:equivalentClass ?VmfChild}
     UNION {?OutTerm owl:equivalentProperty ?VmfChild} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?OutAVS=<marc21AVS#Relationship>) .
filter(?InAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfTerm rdfs:subClassOf ?VmfParent}
     UNION {?VmfTerm rdfs:subPropertyOf ?VmfParent} .
  {?OutTerm owl:equivalentClass ?VmfParent}
     UNION {?OutTerm owl:equivalentProperty ?VmfParent} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?InAVS=<marc21AVS#Relationship>) .
filter(?OutAVS=<onixAVS#CodeList17>)
}

select ?VmfTerm ?InTerm ?InAVS ?OutTerm ?OutAVS
from coa_itd:Graph.Ontology
where {
  {?InTerm owl:equivalentClass ?VmfTerm}
     UNION {?InTerm owl:equivalentProperty ?VmfTerm} .
  ?InTerm vmf:IsInVocabulary ?InAVS .
  {?VmfTerm rdfs:subClassOf ?VmfParent}
     UNION {?VmfTerm rdfs:subPropertyOf ?VmfParent} .
  {?OutTerm owl:equivalentClass ?VmfParent}
     UNION {?OutTerm owl:equivalentProperty ?VmfParent} .
  ?OutTerm vmf:IsInVocabulary ?OutAVS .
filter(?OutAVS=<marc21AVS#Relationship>) .
filter(?InAVS=<onixAVS#CodeList17>)
}
```