The Information Management Framework

Developing Thin Slices

An Introduction to the Methodology for Developing the Foundation Data Model and Reference Data Library of the Information Management Framework

Version – final draft2 – March 2022

Contents

Executive Summary

The report *Managing Shared Data* (West, forthcoming) introduces the 'Thin Slices Methodology' that replaces the 'Corpus Collection' approach described in *The Pathway towards an Information Management Framework*. This report aims to work with data owners to improve information, identify elements of the Foundation Data Model and Reference Data Library and validate the Top-Level Ontology.

This *Developing Thin Slices* report provides a technical description of the process at the heart of the Thin Slices Methodology with the aim of providing a common technical resource for training and guidance in this area. As such it forms part of the wider effort to provide common resources for the development of the Information Management Framework.

It focuses on the process at the core of the Thin Slices Methodology. In particular, it identifies a requirement for a minimal foundation for these kinds of processes. In the companion report, *Top-Level Categories* (Partridge, forthcoming), the foundation adopted by the Information Management Framework is described. Together, the two reports cover the details of the developing thin slices process.

1. Introduction

1.1 Background

In 2017, the National Infrastructure Commission published *Data for the Public Good* (NIC, 2017) which set out a number of recommendations including the development of a UK National Digital Twin supported by an Information Management Framework[1] of standards for sharing infrastructure data, under the guidance of a Digital Framework Task Group set up by the Centre for Digital Built Britain.

Much work has been done following this, but in particular

- A vision of how society can benefit from a UK National Digital Twin is set out in *Flourishing Systems* (Schooling, 2020).
- The direction for the technical standards, guidance and common resources needed as part of the Information Management Framework is set out in *The pathway towards an Information Management Framework* (Hetherington, 2020) and updated in *Managing Shared Data* (West, forthcoming).

In particular they identified the need for:

- A **Foundation Data Model**: a data model that provides the structure and meaning of data incorporating a top-level ontology based on science and engineering principles, enabling it to be extended to support the broadest possible scope consistently.
- A **Reference Data Library**: the classes and properties needed for the UK National Digital Twin that enable different organizations and sectors to describe things consistently.
- An **Integration Architecture**: the technical means, including open-source software, for sharing data securely with authorised users.

1.2 Purpose

This report is aimed at developing an understanding of the finer methodological details of the process of developing thin slices being used by the Information Management Framework of the UK's National Digital Twin.

1.3 Target audience

This report is directed at a technical audience.

[1] In "Data for the Public Good" what we now call the Information Management Framework was called the Digital Framework.

2. Context

The National Infrastructure Commission report *Data for the public good* (NIC, 2017) recommends the creation of a National Digital Twin connecting digital twins across different sectors to give a system of systems view of national infrastructure enabling better decisions for better outcomes in its development and use.

The pathway towards an Information Management Framework: (Hetherington, 2020) recommends the adoption of an Information Management Framework (IMF) that includes a Foundation Data Model (FDM) as a key component**.**

Managing Shared Data (West, forthcoming) describes the seven circles approach – shown graphically in Figure 1 – that is being used to organise the development of the IMF. From an IMF perspective, the focus of this report is on the last four circles: Reference Data Library (RDL), the FDM, the Top-Level Ontology (TLO) and the Core Constructional Ontology (CCO).

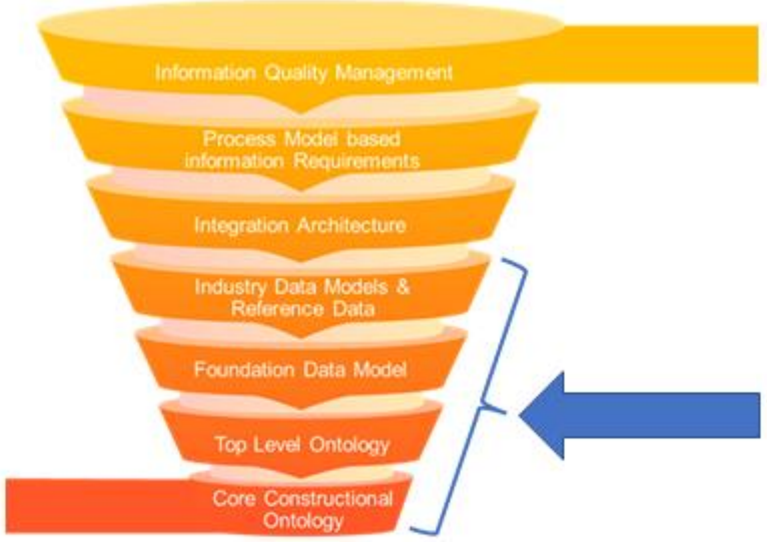
Information Quality Management
Process Model based information Requirements
Integration Architecture
Industry Data Models & Reference Data
Foundation Data Model
Top Level Ontology
Core Constructional Ontology

Figure 1 – Information Management Framework: Seven Circles approach: focus on the last four levels

The Approach to Develop the Foundation Data Model for the Information Management Framework (West, 2020) determined that four selected 4-dimensionalist TLOs – the *selected TLOs* – best met the technical requirements of the FDM. It further recommended that these be used to develop a TLO that is used to found an FDM seed and that seed is underpinned by rigorously established foundations.

Managing Shared Data (West, forthcoming) introduces the 'Thin Slices Methodology' that is being used to facilitate the adoption of the IMF, the development of the FDM and RDL and the validation of the TLO and CCO. At the core of this methodology, is an agile thin slices process. This involves selecting appropriate thin slices from datasets to rigorously validate and refine the proposed foundation – as well as exemplify at the data level what grounded data looks like.

This report outlines the data analysis aspects of the thin slice process. These form an iterative full life-cycle approach; using an agile process guided by a minimal foundation – the FDM Seed. The process mines the ontological content for the FDM, bottom up from existing data sets. It focuses on data quality at scale to ensure that the emerging structures are firmly grounded. The process rigorously validates, cleans and transforms the data enriching the semantics and refactoring it so that it more cleanly and correctly fits under the minimal foundation. The process has been developed to minimise costs and risks – while maintaining quality to maximise benefits.

The process is guided by the minimal foundation. The foundation guiding and directing the process is sufficiently minimal that it supports, but doesn't constrain, the analysis. It is sufficiently rich that it both validates and refines both the foundation and the mined content.

It is iterative. The output of one or more rounds of the process can be a stage in the evolution of the FDM and become inputs to a subsequent round as new data (and so requirements) are taken into account. This staged life-cycle approach accommodates the evolution of the FDM in response to emerging requirements.

The companion report, *Top-Level Categories* (Partridge, forthcoming) describes the details of the top level categories that will form the core of the TLO and exclusively and exhaustively divide all entities. The report describes these categories, which have been synthesised from the selected four ontologies and, using a constructional approach, refined into a unified, complete, comprehensive categorical system. The report *Core Constructional Ontology* (Florio, forthcoming) provides a formalisation, giving it a rigorous foundation.

3. Report Structure

There are three main sections in the body of the document. Details have been relegated to the first five appendices with a glossary in the sixth appendix.

This first section provides a context for the life-cycle process in a top-level ontology approach. It starts by introducing the two key components of a top-level ontology based on a full life-cycle approach: a top-level foundation and a grounding process.

The second section looks at two key architectural dimensions of the process; top-bottom and thick-thin. It motivates the choice of a guided bottom-up approach.

The final section describes the IMF choice of approach in the light of the points raised in the previous two sections. More technical details of the approach are given in The Thin Slice Process and Early Thin Slice Examples. It also introduces the selected process – bCLEARer: technical details of which are given in bCLEARer.

4. A top-level ontological approach's two components

In this section, we explain the need for a grounding process in a top-level ontological approach. The end product of the approach is a top-level foundation – so this is clearly a key component. But equally essential is the life-cycle grounding process that both develops and deploys that foundation. One that mines and migrates the ontological content from existing datasets, uses this to develop the emerging framework of the top-level foundation and then uses the framework to ground the ontology emerging from the datasets.

There appears to be a natural general tendency to focus on the end product and not take much account of its life cycle. In the context of top-level ontologies, this results in a focus upon the top-level foundation and significantly less attention, sometimes no attention, on the grounding process from which this emerges. This is clearly reflected in the current status of work on computational top-level ontologies.

This tendency appears in various disciplines. It is not just a modern phenomenon. For example, something similar has happened in the closely related area of formalisation. As (Dutilh Novaes, 2015) notes, in the context of logic, the process of formalisation (analogous to the grounding process) is often neglected and attention is focused on the formal results. She notes the importance of the former for real life application of the latter. She discusses two historical examples of processes of logical formalisation: Aristotle's syllogistic theory from the "Prior Analytics", and medieval theories of supposition, to both illustrate and illuminate how to formalize logical arguments. There is a similar neglect in work on top-level ontologies, with an almost exclusive focus on the formal top level and little or no work done on the process of formalisation.

Another example is product lifecycle management (PLM). This grew out of the perception that, from a data perspective, while there were many successful deployments of systems, these were "islands of automation" in the wider product life cycle process. PLM aims to integrate the management of the entire lifecycle of a product from its inception through the engineering, design, and manufacture, as well as the service and disposal of manufactured products into a single process. The motivation for this is benefits which include (not in any particular order): shorter time to market, better product quality, reduction in prototyping costs, savings through re-use, a framework for product optimisation, savings in reduction in wastage. It should be clear that these are the benefits that a life cycle approach will bring to most products, including TLOs.

This grounding process is crucial in ensuring that critical information is not lost: particularly where the existing datasets are operational, so lost information could compromise the operations. Typically, the benefits of adopting a top-level ontology are heavily dependent upon the quality of its implementation – particularly with regard to achieving interoperability. Using a grounding process helps to ensure that these benefits materialise. It can also, if undertaken correctly, validate and refine the top-level foundation. Furthermore, regimentation leads to a simpler more reusable process, and so reduces costs. In other words, the regimentation the grounding process provides can radically reduce the cost of adoption of the top-level foundation as well as help ensure the benefits of adoption materialise. Hence it is worth taking great care when architecting the grounding process.

5. Choosing the right architecture for the grounding process

In the previous section we have established the need for a life-cycle grounding process. In this section, we look at two dimensions that should be considered when architecting the grounding process: top-bottom and thick-thin – explaining how these should shape one's choice of architecture for the process.

5.1 Top-bottom dimension

This dimension characterises the process on the basis of where it starts. Simplifying, one can think in terms of the dataset to be grounded having three levels; the top, mid and bottom level. These are often associated with metadata, data schema and data levels. Then the two extremes are the top and bottom levels. If one starts at the top, the process works top-down, if one starts at the bottom, it works bottom-up.

In a top-down deployment, the top-level is developed, then the mid (domain) levels and then finally, the data from existing datasets is migrated. Plainly, as the top and domain schema levels are produced before they are used, this would only be successful if one somehow developed a high degree of confidence that these levels will adequately support the requirements of the domain data. Typically this is done through volume testing late in the project – which identifies the missing requirements.

Whereas a bottom-up deployment would start with the existing data and attempt to mine and migrate the ontological commitments, initially at the mid schema level and then eventually the top-level commitments. Unfortunately, this is not usually feasible as, typically, the ontological commitments are far from clear and so the top-level commitments completely inscrutable. But, if it were possible, one would be very confident the top and domain schema levels supported the data – as they are produced from and so grounded in it.

This suggests an architectural accommodation, a trade-off – a guided bottom-up approach. This aims for a balance where the top-level is sufficiently ontologically rich and complex to guide the analysis effectively, but also sufficiently minimal that it does not hinder or block refinements emerging from the bottom (data) or otherwise render the validation ineffective. One aims to seed the grounding process with a top-level that is sufficient to make the ontological foundations scrutable. This could then guide the ontological mining. One also aims to make this as minimal as possible to maximise the benefits of bottom-up grounding. To be as open as possible to refinement as the lower-level ontological commitments emerge from and are confirmed in the data.

5.11 A shift-left testing perspective on this guided bottom-up approach

Looking at this guided bottom-up approach from the viewpoint of (data) validation and verification helps one see the advantages from a different perspective. One can characterise the architectural choice in terms of aiming to shift the tests as far left (along the project timeline) as practically feasible. In the traditional top-down approach, this testing is delayed until the end and so testing is shifted to the right on the project timeline – see Figure 2. The proposed minimal top-level process is a shift left approach, where adding and migrating the data from the existing datasets one is, in effect, testing. Shifting the testing to the earliest possible moment.

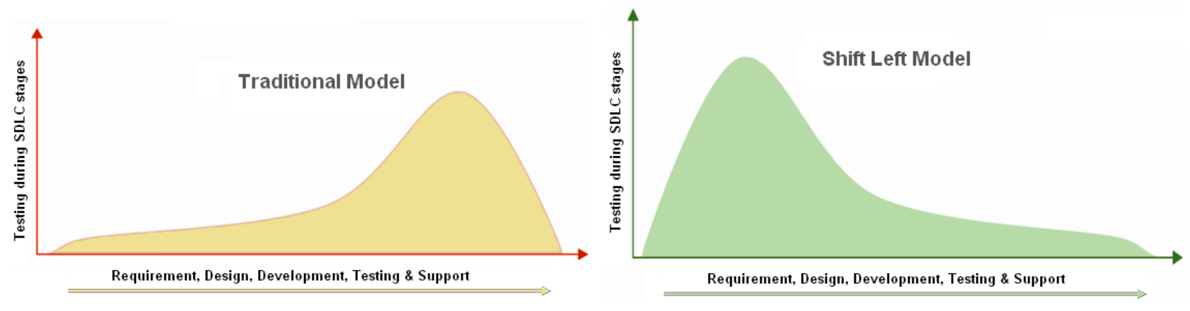

Figure 2 – Typical visualisation of shift left testing

This shift-left testing (Smith, L., 2001) (Bahrs, 2014) (Firesmith, 2015) is a known and respected approach in a number of areas. It is an agile approach much favoured in DevOps, in which one aims to test earlier than usual in the lifecycle. There, it is often described as based upon the first half of the maxim "test early and often”. We have independently arrived at the same architecture for deploying top-level ontologies.

5.2 Thick-thin dimension

Another architectural dimension to consider when deploying is the size of the project. Large projects have well-known risks, hence, other things being equal, smaller self-contained projects are preferable. For most large enterprise systems, the existing datasets are large. So, any project that targets a full data set is likely to be substantial, with all the associated challenges. One of the attractions of the top-down approach is that the first stage of the project – developing the foundation – is relatively small. As already noted, one of the disadvantages is the impossibility of early quality control, that there is little or no serious validation against real data until the end of the project.

Smaller projects have more manageable risks and earlier benefits, including visible improvements in information quality. There are also a range of other benefits, such as a clear sense of progress and the associated increases in motivation and innovation (Amabile, 2011).

6. The IMF approach

In this section we look at how the IMF approach addresses the topics in the two previous sections. With respect to the key two components of a top ontology approach discussed in the first section, the IMF adopts both components of the top-level ontology approach. It clearly recognises the need for a grounding process to develop and deploy its top-level foundation. With respect to the two architectural dimensions of the second section, it adopts a thin, guided bottom-up approach. It encapsulates this in a thin slice methodology.

In the IMF approach, the grounding process is seeded with a minimal top-level and then works from the bottom up, using the seed to guide its analysis. The minimal seeding process enable small projects (such as thin slices of the enterprise) with while also advancing the grounding at the top level – so maximising rather than sacrificing the advantages of early testing and the associated data quality advantages. It also produces early sight of data for the selected domains that has been grounded at the top level. Thus, it is a good mechanism for developing confidence in the approach. The technical details of this approach are in Appendix A, with some early examples, described in Appendix B.

This naturally leads to an extreme shift left as it facilitates testing at the earliest, left-most feasible stage of the development. We have taken care to develop an agile architecture that supports the volatility of this approach, that provides an agile scalable implementation – technical details of this are in Appendix C.

6.1 Grounding process – bCLEARer

The IMF's grounding process has been built upon an established grounding process developed for the selected TLOs. As noted earlier, there appears to be an understandable natural tendency to focus on the top-level foundation and ignore the grounding process from which this emerges. In the TLO world, the IMF's four selected TLOs are an exception to this trend. The earliest TLO, BORO, emerged from work done in legacy modernisation and so the grounding formalisation-migration process was central – the life cycle aspects could not be easily ignored. This life-cycle process has been in continuous development since the late 1980s; its current incarnation is named bCLEARer™.

This makes bCLEARer a natural choice for the IMF (given its choice of TLOs). It is designed to fit with the selected ontologies – and its long history adds a degree of confidence in its operation. Further technical background to this bCLEARer process is given in Appendix D – and to the related topic of levels of digitalisation in Appendix E.

7. Conclusion

We have outlined how FDM is being developed using a thin slice approach; one which is an iterative full life-cycle approach that uses an agile process. This is guided by the top-level categories that provide a rigorous foundation. We have described how the process builds the FDM from the bottom up using existing data sets, how it focuses on data quality at scale to ensure that the emerging structures are firmly grounded. How the process rigorously validates, cleans and transforms the data enriching the semantics and refactoring it so that it cleanly and correctly fits under the top-level categories. How the process has been developed to minimise costs and risks – while maintaining quality to maximise benefits.

Appendix A. The Thin Slice Process

The IMF is adopting a thin slice methodology – as described in *Managing Shared Data* (West, forthcoming). This appendix describes in more detail the process for developing thin slices which is at the heart of this methodology.

Breaking a problem down into thin slices is a common pattern in approaches to transformations, including digital transformation. (For example, one other current similar emerging approach is (O'Brien, 2019). Using a thin slice methodology is designed to be an agile approach to change. It achieves this by being iterative and incremental, aiming to evolve rapidly with each increment. It is structured to cope with change that requires rapid learning. It is capable of working in discovery mode, where one needs to discover the answers. And hence the overall direction only emerges through analysis. In other words, that there are areas where one does not already have the answers and in some cases, nor do others, so one cannot just copy from a successful peer. It has built in processes for these situations, where one advances incrementally based on what one finds works – coming up with hypotheses about the changes one needs to make, testing them, learning from them, and then reviews, refines and repeats. This creates short-term wins that can be built upon. It also means that there is a clear sense that what is produced has been validated as fit for purpose (Note: as this description makes clear, there is only a tenuous relationship to the psychological sense of 'thin slice' (Ambady, 1992).)

The bCLEARer grounding approach (Appendix D) is typically deployed using thin slices. For the IMF, it is intended to deploy it as part of its thin slice methodology as an efficient and effective way of generating a tested top-level ontology, foundation data model and associated reference data. In particular, the use of thin slices is expected to produce feedback in a short space of time. For the development of the IMF model, the thin slices will be targeted on areas where patterns need to be developed or tested for the top-level hierarchy. Early thin slices (Appendix B) have targeted classification and onomatology (names).

A.1 Harmonising multiple thin slices

Typically when adopting a thin slices approach, one of the challenging areas is harmonisation across independent ontologies developed in the thin slices. However, the bCLEARer thin slice approach was developed with the harmonisation requirement centre stage. It starts with the advantage of using a common top-level foundation across the thin slices, which significantly simplifies harmonisation. It also has well-developed, tried and tested, processes for developing a common ontology across multiple heterogeneous datasets. This latter is vital not only for developing a harmonised ontology across systems and domains but also for being able to validate the mined ontology against different sources. Various aspects of this process are described in the papers listed in Table 1.

Year	Title	Relevant Keywords	Link
2019	Coordinate Systems: Level Ascending Ontological Options	semantic modernisation	https://borosolutions.net/coordinate-systems-multi-2019
2013	A Robust Common Master Data Foundation for Oil and Gas	semantic integration	https://borosolutions.net/robust-common-master-data-foundation-oil-gas

Year	Title	Relevant Keywords	Link
2013	Air Control Means: An 'Improving Precision' Case Study	semantic modernisation	https://borosolutions.net/air-control-means-ontobras-2013
2013	Geospatial and Temporal Reference: A Case Study Illustrating (Radical) Refactoring	semantic refactoring	https://borosolutions.net/geospatial-temporal-reference-ontobras-2013
2013	MODEM – Behaviour: A 'Structural Constraints' Case Study	semantic modernisation	https://borosolutions.net/modem-behaviour
2013	Ontology – Introduction: The Industrial Application of Ontology: Driven by a Foundational Ontology	ontology patterns	https://borosolutions.net/ontology-introduction-ontobras-2013
2013	Re-Engineering Data With 4d Ontologies And Graph Databases	semantic modernisation	https://borosolutions.net/reengineering-data-4d-ontologies-graph-databases
2013	Tullow's Master Data	semantic modernisation	https://borosolutions.net/tullows-master-data
2012	DOD Architectures and Systems Engineering Integration	semantic modernisation	https://borosolutions.net/dod-architectures-systems-engineering-integration
2012	Modem – Reengineering the MODAF Meta-Model Based on the IDEAS Foundation Model	semantic modernisation	https://borosolutions.net/modem-reengineering-modaf-based-ideas
2012	Ontology for Big Systems & Systems Engineering	semantic modernisation	https://borosolutions.net/ontology-big-systems-systems-engineering
2011	A Novel Ontological Approach to Semantic Interoperability Between Legacy Air Defense Command and Control Systems	semantic integration	https://borosolutions.net/novel-ontological-semantic-interoperability
2011	Demonstrating A Successful Strategy for Network Enabled Capability	semantic modernisation	https://borosolutions.net/successful-strategy-network-enabled-capability
2011	Semantic Modernisation: Layering, Harvesting and Interoperability	semantic modernisation	https://borosolutions.net/semantic-modernisation

Year	Title	Relevant Keywords	Link
2011	Developing high quality data models (See Ch. 3.2 Integration of Data and Data Models.)	harmonisation	https://www.elsevier.com/books/developing-high-quality-data-models/west/978-0-12-375106-5
2008	Introduction to Ontology – Tutorial	ontology patterns; semantic modernisation	https://borosolutions.net/introduction-ontology-iea-2008
2004	Software Stability: Recovering General Patterns Of Business	semantic modernisation	https://borosolutions.net/software-stability-amcis-2004
2002	The Role of Ontology in Integrating Semantically Heterogeneous Databases	semantic integration	https://borosolutions.net/role-ontology-integrating-heterogeneous-databases
2002	The Role of Ontology in Semantic Integration	semantic integration	https://borosolutions.net/role-ontology-semantic-integration
1996	Business Objects: Re-Engineering for Re-Use (See Ch. 18 Starting a Re-Engineering Project)	semantic modernisation	https://borosolutions.net/business-objects-1st-edition
Forthcoming	Managing Shared Data	thin slices	

Table 1 – Selection of literature related to thin slices

A.2 Managing around thin slices

For enterprise application systems of any size, it is likely that they will decompose into multiple thin slices. From a validation perspective, it makes sense to implement in steps – as each implementation provides evidence of fitness for purpose. This creates a situation much like Otto Neurath's boat, where we "are like sailors who on the open sea must reconstruct their ship but are never able to start afresh from the bottom. Where a beam is taken away a new one must at once be put there, and for this the rest of the ship is used as support. In this way, by using the old beams and driftwood the ship can be shaped entirely anew, but only by gradual reconstruction." (Neurath, 1973).

In these Neurath cases, one is reconstructing the system (ship) in thin slice chunks (planks) much as shown in Figure 3.

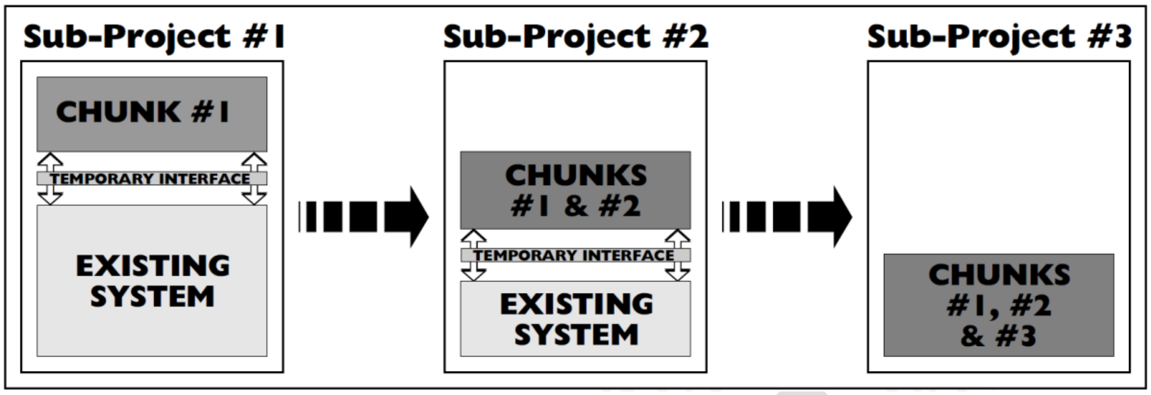

Figure 3 – Rebuilding a system in chunks (based upon Figure 18.7 from (Partridge, 1996))

One of the concerns in this situation is the interconnectedness of the content across the thin slices. As this is likely to lead to cases where the cleansing and grounding in a later thin slice gives rise to a new perspective on earlier thin slices. And this in turn, leads to a requirement to revisit the data from the earlier thin slice.

Where data is reasonably clean and grounded this situation is less likely to occur. Where it is not, it is more likely. But in this case, it is also more likely that the cleansing and grounding leads to significantly conceptually cleaner outcome – as illustrated in Figure 4. This is likely to offset the costs of revisiting earlier slices.

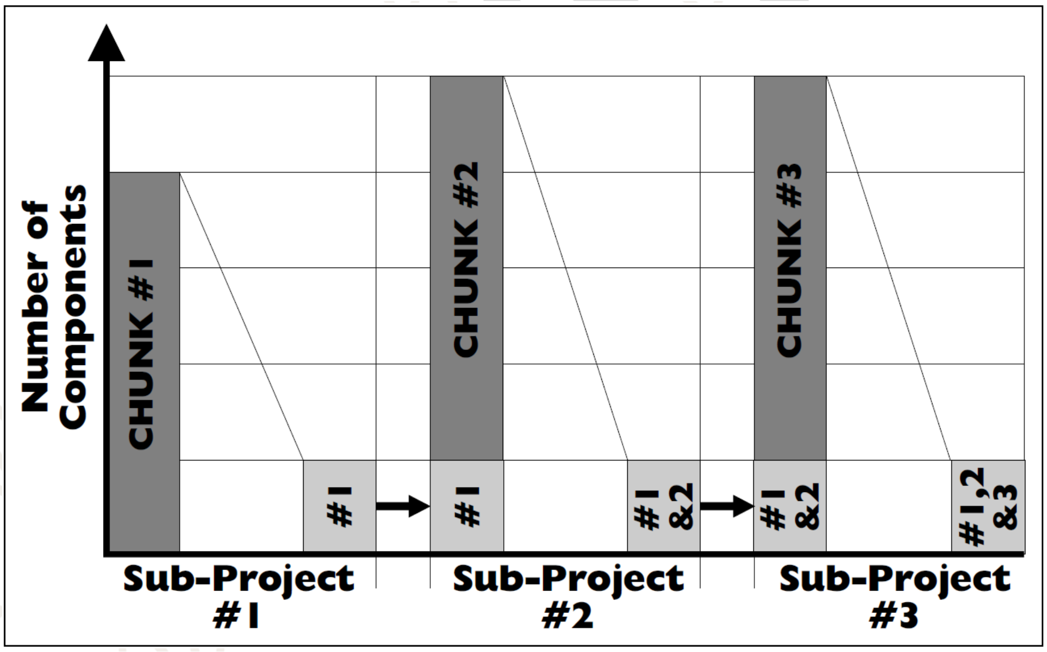

Figure 4 – Simplifying in stages (based upon Figure 18.8 from (Partridge, 1996))

Appendix B. Early Thin Slice Examples

As noted elsewhere, the IMF is adopting a thin slicing methodology – for example in Appendix A which describes the thin slice approach. In this appendix we describe two examples of thin slices: UNICLASS and onomatology.

B.1 Thin Slice: Classification – UNICLASS

Taxonomic classification is a central pattern not just in most computer systems, but in scientific disciplines such as biology. Accordingly, it is a prime topic for a thin slice. UNICLASS is a unified classification system for the construction industry – illustrated in Figure 5. Using it is a requirement for BIM (Building Information Modelling) projects, to comply with BS EN ISO 19650 series of standards. The latest version is UNICLASS 2015 (https://www.thenbs.com/our-tools/uniclass-2015). It was selected as a candidate for a classification thin slice and analysed in 2019 (Partridge, 2020).

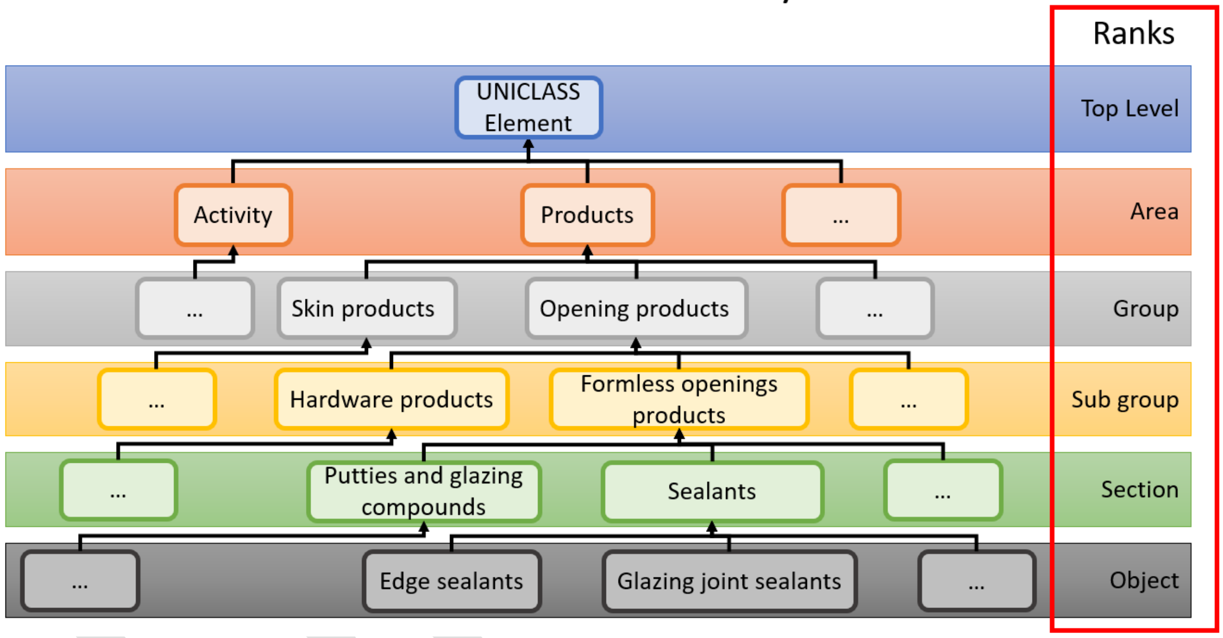

Figure 5 – The UNICLASS Hierarchy

The analysis was aimed at revealing the underlying ontological commitments such a classification structure has. It found two general ontological commitments; higher-order types and first-class relations – establishing these as a requirement for any top-level ontology that includes the UNICLASS classification.

An informal description of the analysis is available in (Partridge, 2020)**.** The formal description of the analysis is the Python code available on GitHub here: https://github.com/boro-alpha/uniclass_to_nf_ea_com. The analysis used a simplified top-level category system that is visible in the GitHub data. This is a good example of how thin slices can reuse patterns discovered earlier. In this case, the patterns were originally raised in (Partridge, 1996) 'Chapter 7 – Physical Bodies as Four-Dimensional Objects – Section 5 – Classes of four-dimensional objects' and most recently summarised in (Partridge, 2016).

B.2 Thin Slice: Onomatology – Nomenclature

Onomatology is the study of the etymology, history, and use of proper names. Proper names are those that uniquely identify the thing in the world they name. Examples include names of people (such as Joe Biden) or a place (London, England). Things often have many proper names; London is also known as Londres (in French) and Lontoo (in Finnish).

In many ontologies, indeed in many computer applications, objects will have a single (proper) name. While this may simplify things for the developer, it is not operationally practical as in most domains there will be a number of names for salient objects. As the four-dimensional ontologies were initially developed in the context of legacy modernisation, the requirement for multiple names was recognised as key from the start. A four-dimensional naming pattern that placed no constraints on the number of names was developed for four dimensional ontologies. It was originally described in some detail in (Partridge, 1996) and subsequently incorporated in the other selected TLO's – for example (West, 2011). The most recent paper on this topic is (Partridge, 2019), where nomenclatures and the notion of name exemplar are examined in detail.

The ubiquity of the name pattern makes it a good target for a thin slice. The Ordnance Survey (OS) produce open data with structures based upon the INSPIRE Generic Conceptual Model (https://inspire.ec.europa.eu/documents/inspire-generic-conceptual-model) for spatial data infrastructure. One of the datasets (OS Open Names – https://www.ordnancesurvey.co.uk/business-government/products/open-map-names) is a good candidate for a number of thin slices – as shown in Figure 6.

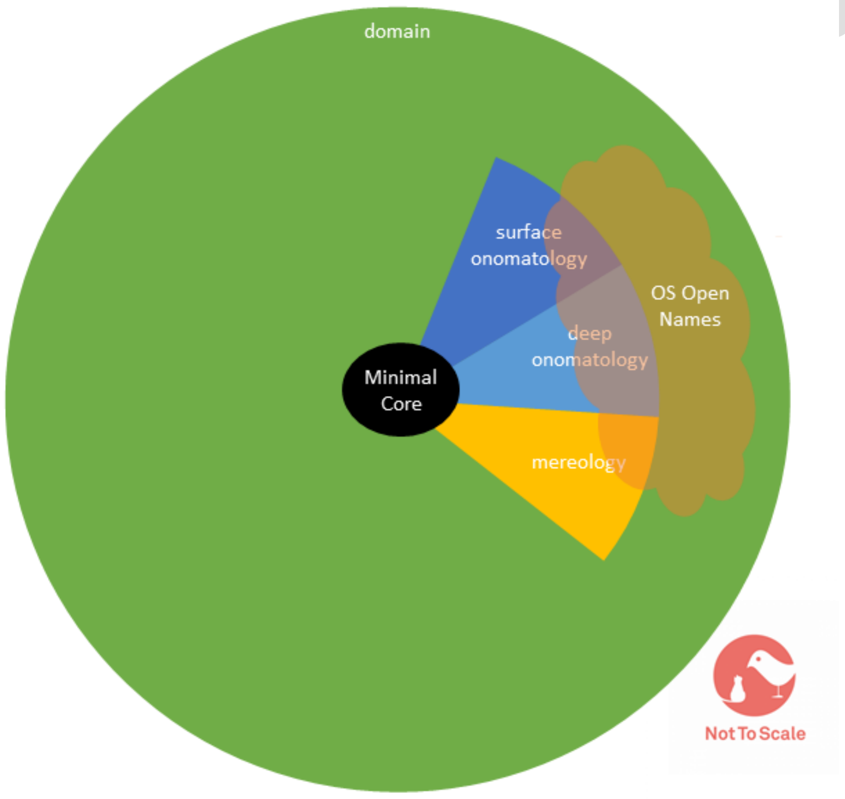

Figure 6 – OS Open names – thin slice candidates

The onomatology work has been divided into two; surface and deep onomatology. Surface onomatology, as the name suggests work with objects and names where there is a reasonably clear distinction as to which one is which. Deep onomatology deals with cases where objects and names are tightly, closely entwined with a much less clear distinction between them. The surface onomatology work is complete and the open-source output is stored here: https://github.com/boro-alpha/bclearer_boson_1_1.

One clear lesson from the onomatology work is that it is rarely a good idea to adopt a naming pattern that restricts objects to a single name.

B.3 Summary

These two examples provide good examples of many of the features of thin slices. The open-source code, particularly, provides a clear example of how the migration can (and should) be automated.

Appendix C. Agile Ontology Oriented Architectural Styles

The thin slice process brings with it a requirement for a language (a way of storing the semantics – representing the domain) with an architecture that is both agile and ontology-oriented. The language needs to be sufficiently flexible (agile) at scale. It also needs to be able to handle the semantic volatility that arises during the various iterations (evolutions) of the domain dataset as it is migrated and transformed. And it needs to be able to accommodate the top-level category system in the final iterations. Given the adoption of shift-left testing inherent in the approach, the ontology will be dataset-size, so the language also needs to be able to scale to multiple enterprise datasets-size.

To meet this need for ontological agility at scale, as explained below, we have found it makes practical sense to adopt languages whose architectural style pays particular attention to their semantics and how this is related to the syntax. The term 'language' is used here in the sense of a medium that is used to store (and so represent) the ontology. Hence, this includes both databases (for example, relational and document databases) as well as what are more traditionally called 'computer languages' (such as Java and JSON).

C.1 Semantics for languages with a stratified 'metamodel' syntax

Many computer languages (in the general sense above) have an architecture style whose syntax is stratified into a sequence of levels, that are ordered by a 'metamodel' relation. Where a metamodel is a model that consists of statements about models: its universe of discourse is a set of models and its statements are about the constructs in those models. In a similar way to models being about a domain (some portion of reality), so metamodels are about models – their domain is a model. Similarly, meta metamodels are about metamodels, they are models of metamodels containing statements about metamodels.

C.11 Stratified relational-type databases

A good example of this style is relational-type databases that organise data into rows and columns in a series of tables. This can be stratified into three levels

1. Category (metamodel) – the types; tables, columns, rows and cells.
2. Schema (model) – specific tables and columns.
3. Data – the rows and cells in the specific tables.

This style has been common in computing for a while, see for example Abrial's (Abrial, 1974) analysis of a binary data model into three levels – the data level of a database, the schema of the database (model), and the category level (metamodel).

The current common architectural style for overlaying semantics onto this stratified syntax is to follow the stratification. Each stratum has its own sub-semantics, where it refers to a restricted subset of the objects in the domain. This divides the domain into stratified levels – based on the order of its types. Objects in a lower level are instances of types at the next level. For example, a common (usually informal) semantics maps the general type 'table' onto the general entity-type object and the specific table 'customer' is mapped onto the specific entity type 'customer' – which is an instance of (and so one type level lower than) the general entity-type and so on. One should note that this architectural style prioritises the 'instantiation' relation as the single basis for stratification.

C.12 UML's Stratified Meta-Object Facility (MOF)

UML's Meta-Object Facility (MOF) (OMG, 2016) is in the same tradition using a similar stratified syntax – though it has relaxed the height constraint. There are recursive strata (named M0, M1, M2, etc), where each higher level provides the syntactic metamodel for the level it succeeds; M2 is metadata for M1. And so the higher level contains types whose instances are in the lower level – see Figure 7.

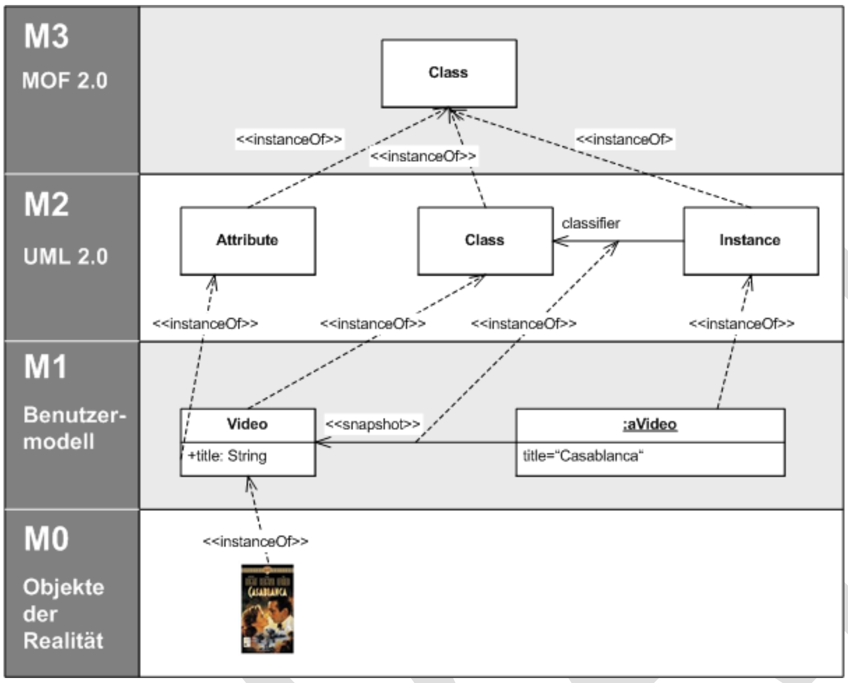

Figure 7 – UML's Model Levels

As noted in (OMG, 2016, sec. 3) this allows for any (finite) required number of levels: "MOF 1 and MOF 2 allow any number of layers greater than or equal to 2. (The minimum number of layers is two so we can represent and navigate from a class to its instance and vice versa)." As shown in Figure 7, the intended semantics is divided between the layers; where particulars are at M0, first order classes at M1, second order classes at M2 and so on. (As noted in, for example, (Partridge, 2020), in this semantic scheme, there is no place for mixed classes, one that contain members of different levels.)

C.13 The stratifying architectural style

There are superficial attractions to this stratifying architectural style for semantics, where the same language structure is used for both syntactic and semantic concerns. But it also places quite a stringent demand on that structure: it becomes important that the structure is adequate for both concerns. This becomes tougher when a project involves the rich and complex top-level semantic concerns and the volatility of evolving domains. In particular, one needs to be sure that the way this architectural style prioritises the 'instantiation' relation as the single basis for stratification is not a ruinous constraint.

If one does not wish to develop one's own language and so is considering using an existing one, then most existing languages have not been developed with a top-level ontology, or evolving domains, in mind, so this is likely to be a big ask. There are two further points to consider when contemplating adopting this style – described below.

Rationalising the language's syntactic structure
A common manoeuvre (in our view a 'mistake') when adopting the stratifying architectural style is, having selected a language, rationalising its explicit syntactic structure as a genuine semantic structure. Within the Information Systems community, one can see this clearly with relational databases, where the table-column structure is rationalised as an entity-attribute ontology – this (and the associated difficulties it raises) is described in detail in (Partridge, 1996, Chapter 3)**.**

The temptation to make this manoeuvre is not restricted to computing. There is a similar rationalisation manoeuvre and analogous unease within philosophy, where there has long been a recognition that the syntactic structure of natural language is not a totally reliable guide to the underlying semantic (and so, ontological) structure. For example, E. J. Lowe (Lowe, 2006) says "In point of fact, I do not at all think that metaphysics should be conducted entirely through the filter of language, as though syntax and semantics were our only guides in matters metaphysical—although it should hardly be surprising if natural language does reflect in its structure certain structural features of the reality which it has evolved to express." And referring to (Sommers, 1983): "it is wrong-headed to attempt to make our logical syntax conform to our ontological preconceptions and, indeed, that some of these preconceptions may in any case be rooted in antecedently assumed syntactical distinctions which possess little merit in themselves.". This tension is ancient. J. L. Ackrill suggests Aristotle's *Categories* "is not primarily or explicitly about names, but about the things that names signify … Aristotle relies greatly on linguistic facts and tests, but his aim is to discover truths about non-linguistic items" (Ackrill, 1963, p. 71). Whereas Baumer (Baumer, 1993, sec. Abstract) proposes that "the *Categories* sets forth a theory of lexical structure".

Kit Fine (Fine, 2017) makes the first step towards another, less contentious, way of rationalising natural language that exposes the concern. He distinguishes between the *naive metaphysics* that is reflected in language and *foundational metaphysics* that aims to reflect what there really is. Friederike Moltmann (Moltmann, 2017) takes this rationalisation a step further by introducing *Natural Language Ontology.* This aims to reflect the structures that are implicit in the use of natural language – not the world. So this rationalisation is clear about its nature (unlike the relational database rationalisations mentioned earlier) it does not aim to be a foundational metaphysics, an account of the real world.

For us, the lesson to learn from this (following Fine's path to *foundational metaphysics*), is that if one wants to 'reflect what there really is' then it makes more sense to provide the tools to do this than uncritically adopt the syntactic structure of a language designed for other purposes.

Aligning the syntactic strata with the life cycle stages
Along with the 'aligning the semantics with the syntactic strata' architectural style, there is a traditional sub-style that aligns both of these with life cycle stages. Simplifying a little, one can divide the life cycle into three stages of production:

1. framework – producing the general tools such as programming languages and databases,
2. application – producing the application using framework tools – and
3. operation – producing the data that populates the application.

Traditionally, there is an architectural sub-style: 'align life cycle with syntax (and so semantics)', where the three concerns follow the same strata – as shown in Figure 8. As is clearly visible, this aligns the semantics with the syntactic strata.

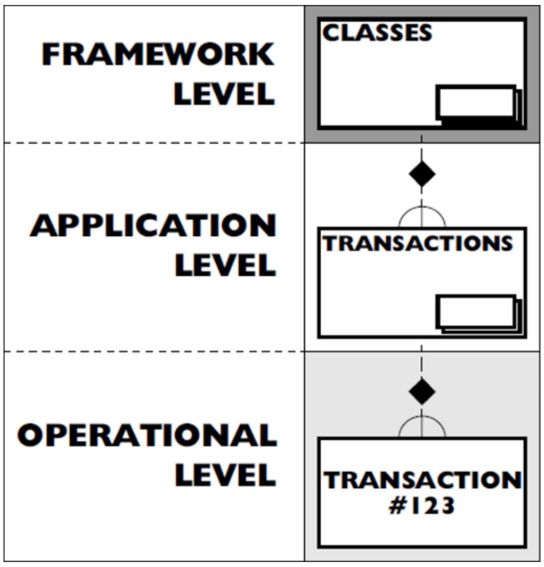

Figure 8 – The three life cycle stages (based upon Figure 11.6 (Partridge, 1996)

If one views this from the perspective of the requirement for a language that is agile at scale, one notices that the architecture constrains agility across the strata. So, in principle, the framework becomes rigid when the application is being produced (though the application elements may be agile) – similarly the application becomes rigid when the operational data is being produced (though the operational elements may be agile). There are many good aspects of this way of working. Technical concerns are resolved at one stage level and the solutions are immutable later stages, facilitating reuse. However, as it stands, the approach gives the operational stage little governance over semantic decisions made at the earlier stages – or the application stage over the earlier framework stage.

When one is faced with rich and evolving semantics that need to be reflected in the semantic structure, then one typically needs governance over all the semantic choices. So this style is likely to severely hamper agility.

C.2 A different architectural style for a stratified syntax – 'integrated semantics'

The previous section looked at a style where the semantics is divided to follow the syntactic stratification, and noted how it constrained agility. In this section we look at the other style, where it is not divided; where there is what we could call an 'integrated semantics' at a single level. We show how this does not constrain agility. This shows that one can, very broadly, distinguish and so choose whether or not to use an architectural style based upon where the semantics follows a syntactic stratification – so whether or not to constrain agility.

To illustrate this, let's return to the earlier three level relational-type database example. In this case, the metamodel and model play a purely syntactic role, and all the semantic work is done at the data level. Only signs at this level refer to the domain. So, the signs that refer to the general entity-type and its instance the customer type appear at this level. Objects in the ontology (domain) are represented directly in the data – it has a single container. There is not a stratified semantics and so there is no prioritisation of one part of the semantics, the 'instantiation' relation, as the single basis for the architectural structure.

One way of thinking of this second style is there being two related domain sub-ontologies. The first is the salient ontology of the domain being modelled (the semantic concern), the second the less salient ontology of the (domain of the) representations (the signs) from which the model is constructed (the syntactic concern). In this case, there is a separation, where the metamodel and model work with the syntactic concern and the data with the semantic concern.

For more detail see (Partridge, 1996, Chapter 9) which describes these two domains, including how they are tightly bound by referring relations. This can then be fitted into a single model, as Figure 9 shows, and so one can then regard the two sub-ontologies as fitting under a common top-level ontology.

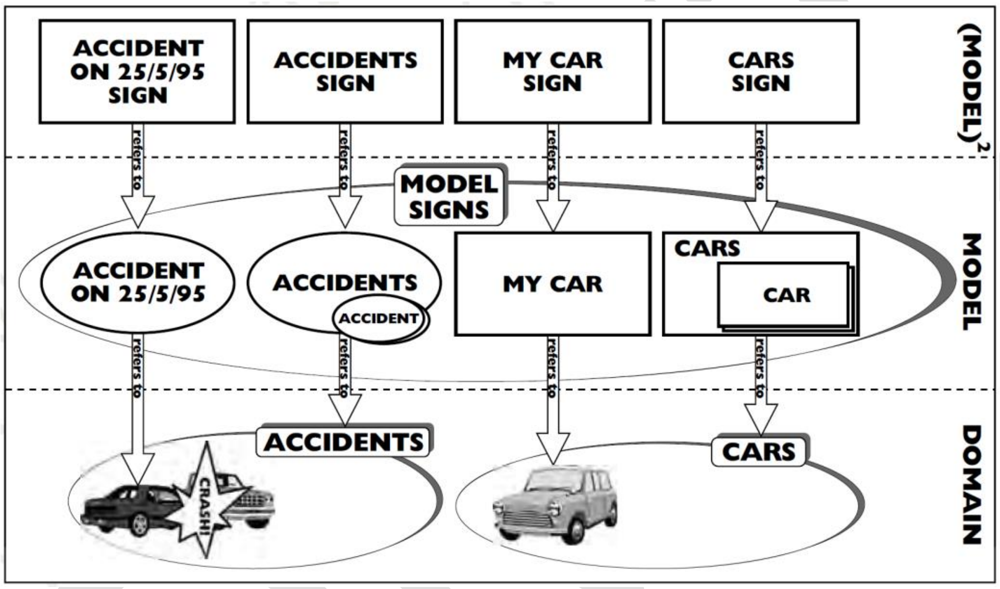

Figure 9 – Modelling the model (from Figure 9.39 (Partridge, 1996))

Adopting a similar style in UML only requires two model levels; syntactic and semantic. The syntactic model describes the signs – it is about the boxes and lines drawn in the semantic model (or their data equivalent), not what they represent. It may have some implied semantics, in the sense that what one chooses to represent by a box rather than a line may reflect the domain semantics. All the objects in the ontology are represented by the semantic model.

This is not a novel approach. In the early days of database computing, the 1970s and 1980s, there were many examples of this style of building systems (the authors came across some in their early days working in the computer industry). The databases used in the grounding analysis in (Partridge, 1996) are an example. One public example is the KALIDO Active Information Management software system developed by Shell and later spun off into a separate company to market as a software product. There is a connection with the NDT IMF work in that this used a generic model that was based upon an early version of the EPISTLE model (which subsequently evolved into one of the NDT's selected TLOs – ISO 15926). Practitioners have provided a wealth of anecdotal evidence of the extreme resilience of this approach in the face of the large-scale volatility of the evolving domain models. This kind of resilience is required for the thin slice bCLEARer work to help accelerate the evolution of the domain models.

C.3 Examples from other related areas

The trade-offs between these architectural styles are based upon general structural features and so appear in a number of other areas; we outline some below.

C.31 Adaptive object model

In the Object-Oriented world, there is an architectural style, called the adaptive object model (Yoder, 1998). Though it is not described exactly in terms of semantics and stratification, there is a similar movement towards keeping all the semantics in one place under a common governance.

"We have noticed a common architecture in many systems that emphasize flexibility and run-time configuration. In these systems, business rules are stored externally to the program such as in a database or XML files. The object model that the user cares about is part of the database, and the object model of the code is just an interpreter of the users' object model. We call these systems "Adaptive Object-Models", because the users' object model is interpreted at runtime and can be changed with immediate (but controlled) effects on the system interpreting it. The real power in Adaptive Object- Models is that the definition of a domain model and rules for its integrity can be configured by domain experts external to the execution of the program. These systems are important when flexibility and dynamic runtime configuration is needed …" (Yoder, 2002). The developers of this approach are articulate about its benefits. They clearly describe how it gives users the power to design the semantic type structure and so brings flexibility in the face of changing requirements. They note it is common: "Recently I have seen many examples of a type of architecture that was new to me. Half of the demonstrations at OOPSLA'97 were examples of this architecture." (Johnson, 1998).

The focus is on using this at the domain rather than the top-level (which is not really considered). But one can easily shift the focus to top-level ontologies, enabling users to build structures that accurately reflect the top-level of their ontology. There are other similarities: refactoring patterns are supplied to aid the migration to their style, working is a similar way to the grounding process (described in the main body of the paper).

It is clear that in this approach the types and their instances are stored at the object (stratum) level, which does the semantic work of referring to the domain. But there is one quirk. Figure 10 shows the classes in the schema stratum, and can be regarded as the generic (or top-level) model. As such it has a semantics, so there is a vestigial semantics in the schema stratum; not all the semantics is embedded in the object (stratum) level. If one were more interested in dynamic top-level ontologies, the next step would be adding these classes as objects, making the shift of semantics to the object stratum complete.

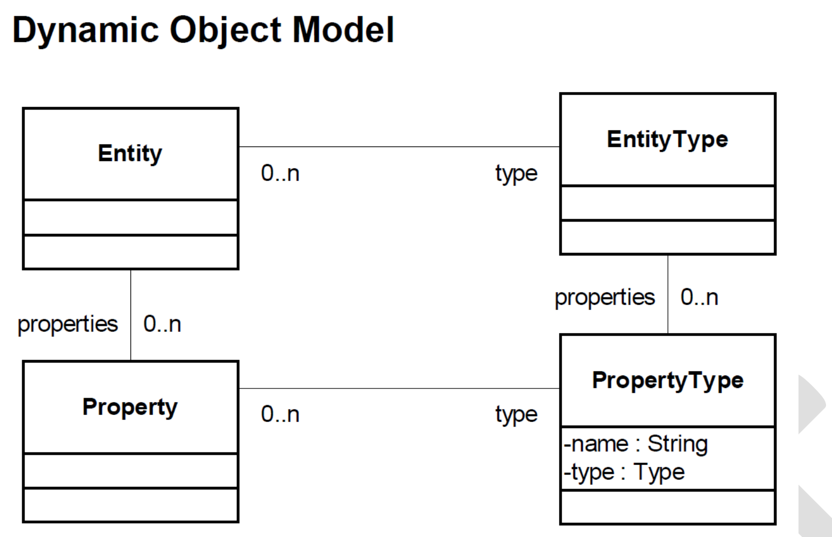

Figure 10 – Dynamic object model (based upon (Johnson, 1998))

C.32 Language-oriented programming

Another similar approach is language-oriented programming (Ward, 1994), which returns control of the semantics to the builders by providing an environment for designing a language. Examples are JetBrains' MPS (Meta Programming System) and open-source Xtext.

These are typically focused at the domain level: "The approach starts by developing a formally specified, domain-oriented, very high-level language which is designed to be well-suited to developing "this kind of program"." (Ward, 1994, p. 1). Hence, these are often used as a software framework for developing domain-specific languages (DSLs), which can then be used in that domain. However, the principles are clearly transferable to top-level ontologies. For example, Xtext has been used (in the background) in the formalisation of top-level ontologies – (Fonseca, 2021).

A clear pragmatic case for their adoption is made: "The approach is claimed to have advantages for domain analysis, rapid prototyping, maintenance, portability, user-enhanceable systems, reuse of development work, while also providing high development productivity." (Ward, 1994, p. 1). And: "Our experience … suggests that there is another large factor of productivity gain to be achieved by developing a suitable *problem oriented* very high level programming language, and using this language to implement the software system." (Ward, 1994, p. 5).

Of course, Language Oriented Programming is more about the management of the process of development than the underlying data structures. Whether these consolidate the semantics at the data level will depend upon the specifics of the design. However, it is quite clear in its aim of enabling a more dynamic approach to domain schemas.

One can regard both this and adaptive object models as a rediscovery of the agility that comes from governance of the semantic structures to the user initially found in the simpler integrated semantics developed in the 1970s and 1980s (mentioned earlier).

C.33 Predicate logic

There is an analogous discussion in the philosophy of predicate logic, one which harks back to the earlier comments about rationalisation. Predicate logic separates predicates from the objects in the domain – so a three-level language that has similarities to relational databases and other metadata-data structures. Many philosophers, including Barry Smith and Jonathon Lowe warn about a rationalisation of predicate logic, where the structure of the logic (language) is assumed to reflect the ontological structure [(Smith, B., 2005) (Lowe, 2012) and (Lowe, 2013) **)**]. Smith has made a clear and spirited attack on this stance which he calls 'Fantology' (Smith, B., 2005, p. 1). Whose description starts: "this is a doctrine to the effect that the key to the correct understanding of reality is captured syntactically in the 'F*a*' … of standard first- order predicate logic. Here 'F' stands for what is general in reality and '*a*' for what is individual. Hence "f(a)ntology". Because predicate logic has exactly two syntactically different kinds of referring expressions—'F', 'G', 'R', etc., and 'a', 'b', 'c', etc.—so reality must consist of exactly two correspondingly different kinds of entity: the general (properties, concepts) and the particular (things, objects)." It then proceeds to describe how this stance arose in modern philosophy, with a cast of, if not villains, then culprits. It ends making the point that: "Our fundamental idea is that predicates (the standard predicates of first-order logic fantologically conceived) do not represent ['representing' here means of course, representing an object in the domain]. … Rather they are what link together variable and constant terms which are those parts of the syntax which do stand for something." (Smith, B., 2005, pp. 19–20). There is a clear similarity with the integrated semantics discussed earlier.

C.4 The thin slice approach

The thin slice approach has a clear requirement to be agile at scale. So it, understandably adopts an integrated semantics architectural style. One which gives the users strong governance over the semantics. Another practical advantage of the approach is that it can, to a large extent, be implementation neutral. Evidence of this is that bCLEARer processes, including the top-level category systems, have been implemented in many traditional structures including: Python, C#, SQL Server, MS Access, JSON and XML.

Appendix D. bCLEARer

The IMF is adopting a thin slice methodology based upon the grounding approach used in bCLEARer. This appendix gives a brief technical description of the bCLEARer process.

The grounding approach is an agile iterative process that is designed to scale. It is a (reusable) component-based, automated process for mining and evolving the ontologies in datasets under a minimal top-level ontology.

It is, from one perspective, an agile approach to change. It progresses in iterations and increments based on what it finds works. It is capable of operating in in discovery mode. It provides a structured way to come up with hypotheses about the changes one needs to make, how to test and learn from them. It aims to learn rapidly with using a regimented and systematic approach which clearly reviews, refines and repeats. This enables it to evolve rapidly at scale in a way that creates short-term wins and builds upon these.

D.1 Background: bCLEARer history

The origins of the bCLEARer process are in legacy modernisation work that started in the late 1980s (both the origins (in the Preface) and the process (in Part 6) are described in (Partridge, 1996). It has been in continuous development since then. Initially the focus was on producing a reliable, repeatable systematic process. In the last couple of decades, the focus has shifted towards producing an automated scalable process – and this revised process was renamed bCLEARer. More recently, the process has been enhanced to measure the digital transformation; the increases in the level of digitalisation (see Appendix E)

The name bCLEARer is an acronym; **C**ollect, **L**oad, **E**volve, **A**ssimilate, **R**euse. Where these are the names of the stages in the process. One can see these stages as corresponding to levels of semantic maturity – as shown in Figure 11.

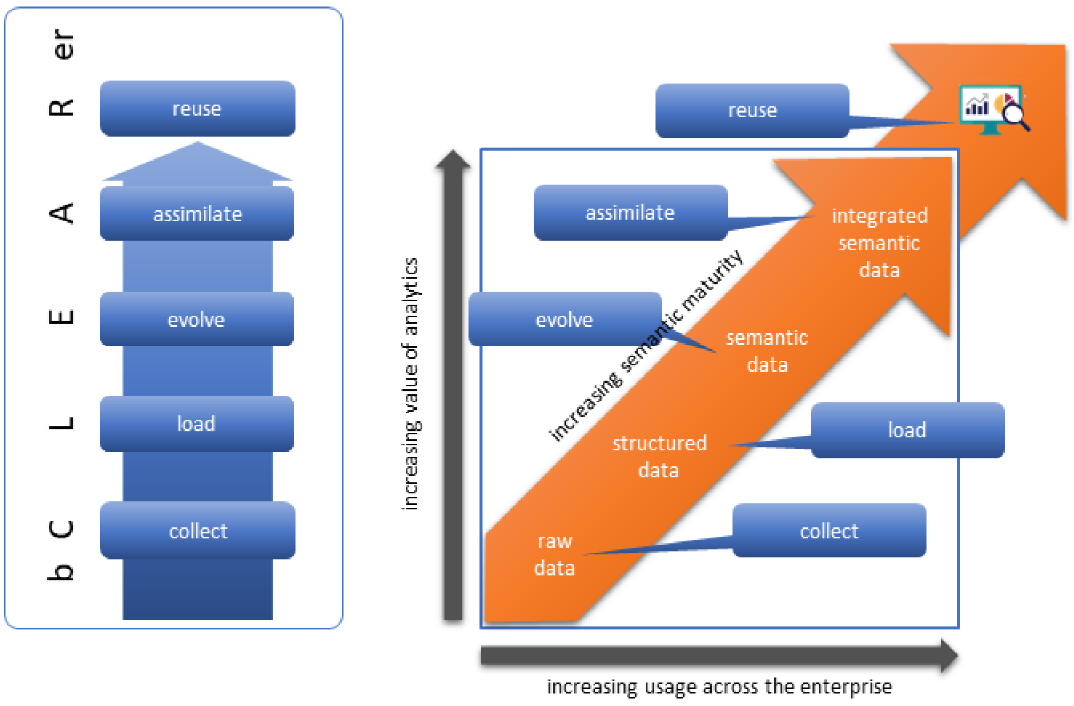

Figure 11 – Mapping onto levels of semantic maturity

D.2 The process

It is an agile, iterative process as shown graphically in Figure 12 and more schematically in Figure 13.

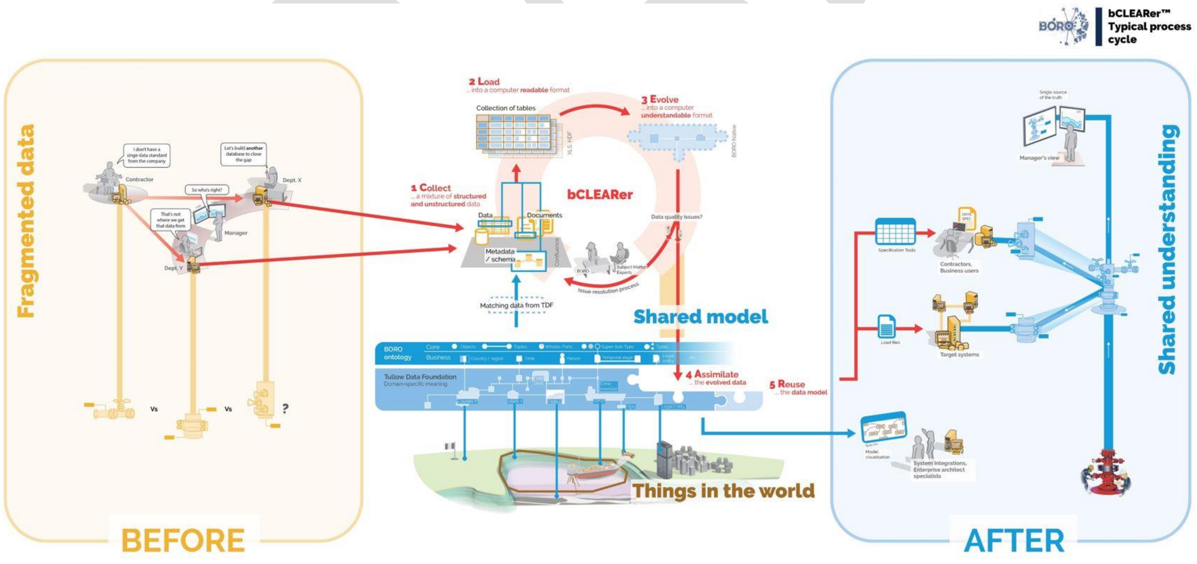
BORO
bCLEARer™ Typical process cycle
Fragmented data
BEFORE
1 Collect ... a mixture of structured and unstructured data
2 Load ... into a computer readable format
3 Evolve ... into a computer understandable format
4 Assimilate ... the evolved data
5 Reuse ... the data model
bCLEARer
Shared model
Things in the world
Shared understanding
AFTER

Figure 12 – A graphical view of the iterative bCLEARer™ Approach

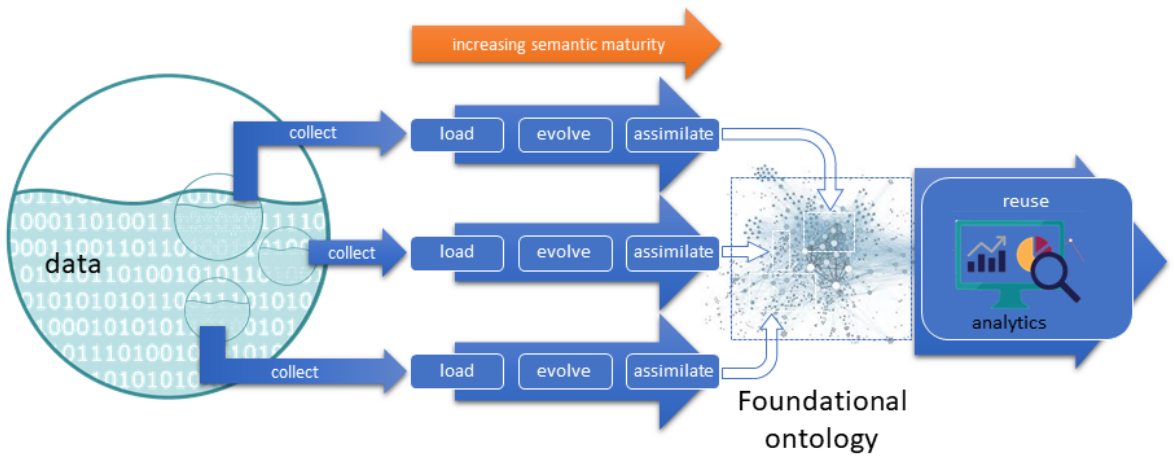

Figure 13 – A schematic view of the iterative bCLEARer™ Approach

These agile iterations can flow in a number of different ways, as shown in Figure 14.

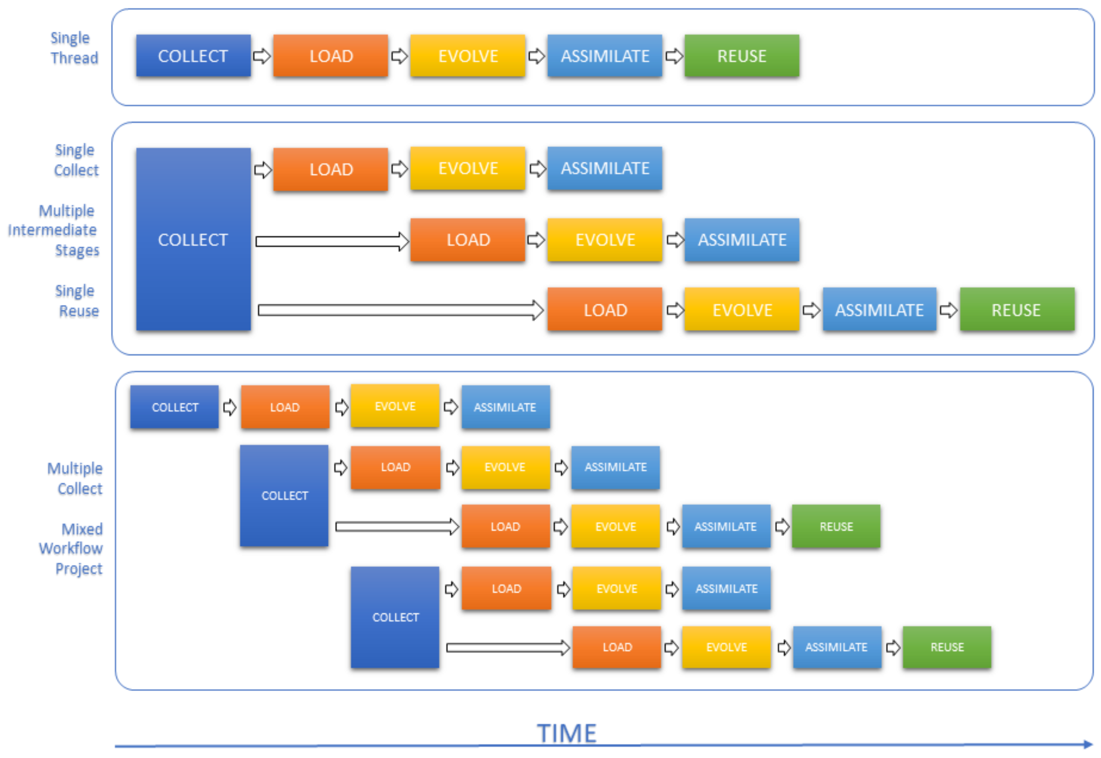

Figure 14 – Typical bCLEARer flows

Improvements emerge throughout the Evolve stage, and for some thin slice projects truncating Evolve and harvesting the improvements is a sensible option. There are natural breaking points in the process – illustrated in Figure 15 and Figure 16 – such as entification. The UNICLASS thin slice described in Appendix B is an example where the Evolve was truncated at the entification stage.

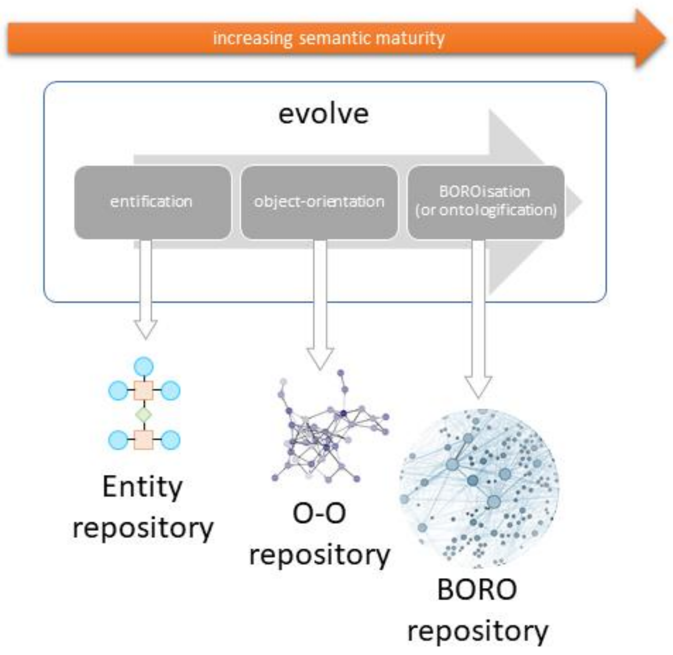

Figure 15 – Natural truncation points in the bCLEARer process

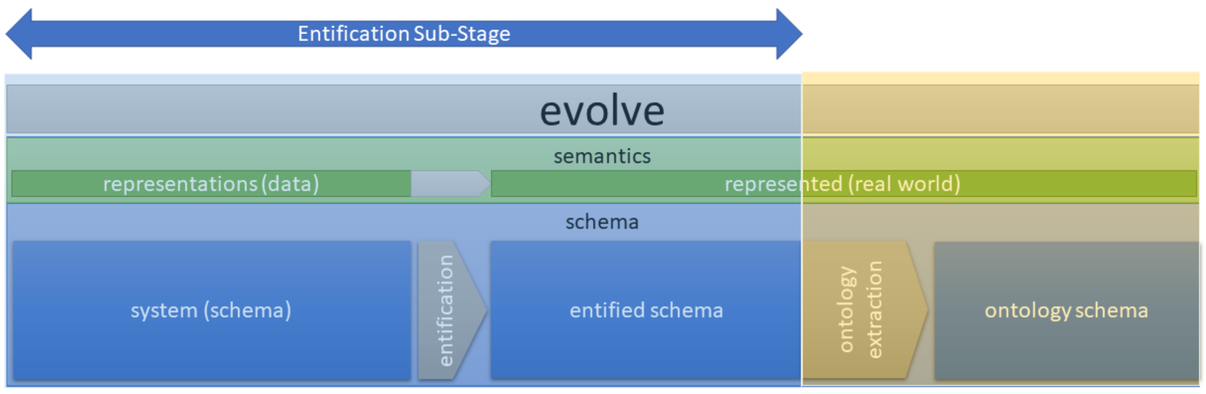

Figure 16 – Anatomy of the bCLEARer process – entification sub-stage

It is good practice to include a number of datasets from different sources (systems) in a thin slice – to provide conceptual triangulation of the data. This raises the requirement to semantically integrate them. A natural architectural style for this is to separate the schema or language in which the data is held from the content and handle these concerns separately. The anatomy for a typical multi-system bCLEARer thin slice process is shown in Figure 17

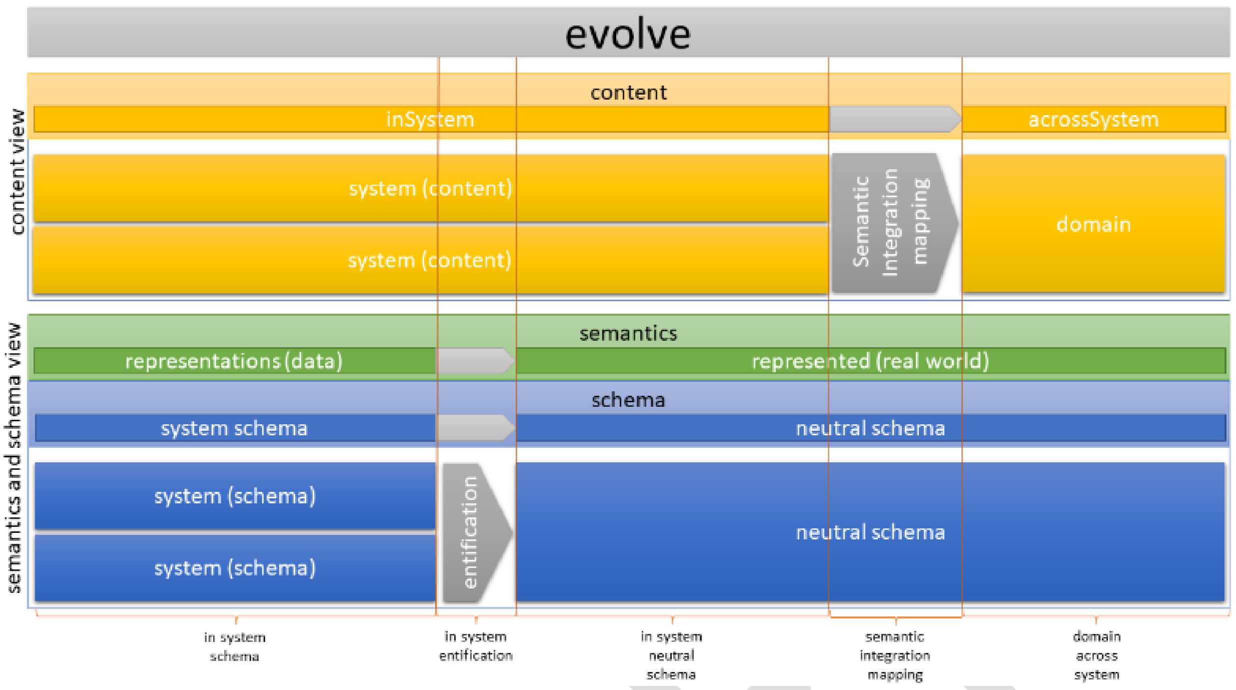

Figure 17 – Anatomy of a typical multi-system bCLEARer process

The process has been designed to be ‘show rather than tell’ approach so that it provides substantive evidence of the efficacy of the approach (in the general spirit of shift-left testing). The process is designed to have simple components, to make it both resilient and easier to implement and understand. It is automated so repeatable and scalable. In the current iteration, this uses Python code. The core knowledge is captured in this code, which (for the IMF project) is being released as open source; this should facilitate knowledge transfer. It is designed to provide visible output at each stage, making the evolution of the data more transparent.

This approach leads to significant reductions in cost and timescales of exchanging dataset by increasing their capability for interoperability. It also leads to substantial corporate benefits such as increases in productivity, resilience and agility (shown graphically in Figure 18).

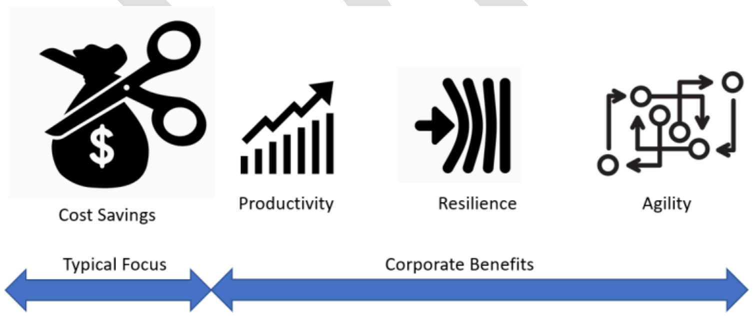

Figure 18 – Costs and benefits

Appendix E. – Digital Transformation: Levels of Digitalisation

A useful benefit of the bCLEARer grounding process (and other similar processes) is its capability to evolve datasets to high levels of digitalisation. This appendix provides a brief technical description of levels of digitalisation.

The IMF team have developed a scheme for assessing the level of digitalisation. This works, like the bCLEARer process at the granularity of data item. This allows us to, among other things, track the digital transformation – the increase in the level of digitalisation – produced by the process. The scheme has been designed with an emphasis on ease of application and use rather than high levels of accuracy. This enables quick and broad assessments.

The scheme has emerged from decades of experience using the bCLEARer method. It is based upon the four facets – the dark blue boxes in Figure 19.

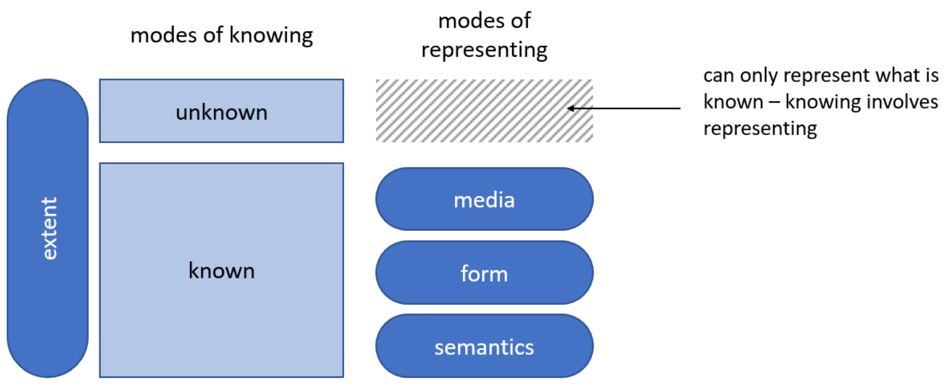

Figure 19 – The four (dark blue) facets

Though these facets are relatively independent, for assessment purposes it is simpler if they are flattened into a sequence of eight levels – as shown in Figure 20.

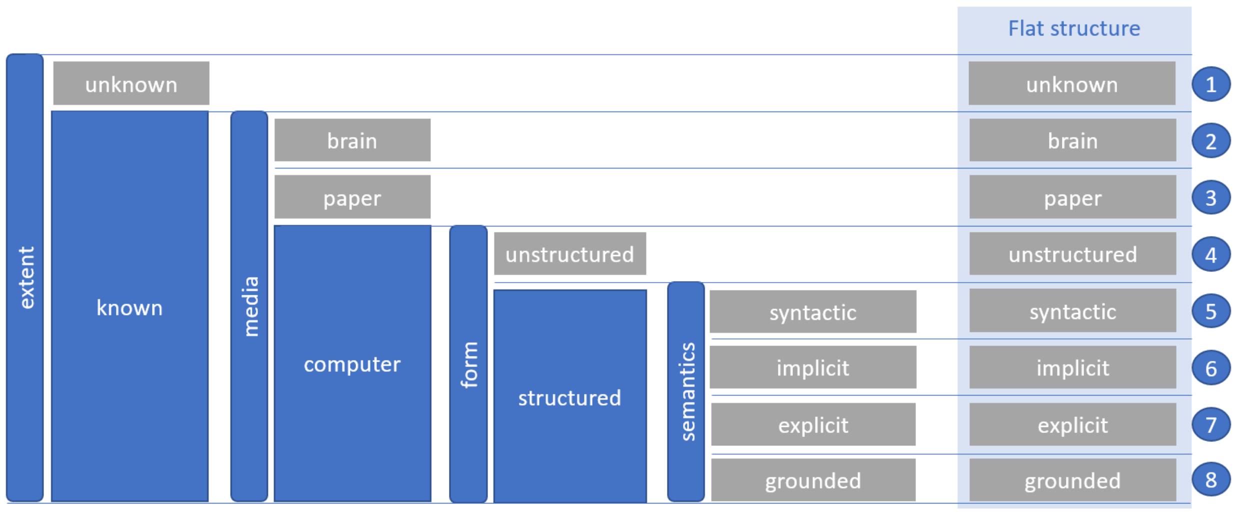

Figure 20 – Flattening the facets

Figure 21 provides a brief description of each flattened level.

	Levels	
1	**unknown**	is not known about at the start of the analysis
2	**brain**	is known but not documented
3	**paper**	is documented, but not on a computer
4	**unstructured**	is not organized in pre-defined manner
5	**syntactic**	is on a computer, structured but no obvious semantics
6	**implicit**	has a semantics one can work out, but does not map directly
7	**explicit**	has a semantics that maps directly to the real world
8	**grounded**	has a semantics that is grounded on a top ontology

Figure 21 – Brief description of each flattened digitalisation level

Typically, the process can be seen as evolving systems of data items from one state to another – as shown in Figure 22. Data items may be transformed as well as implicit items being made explicit.

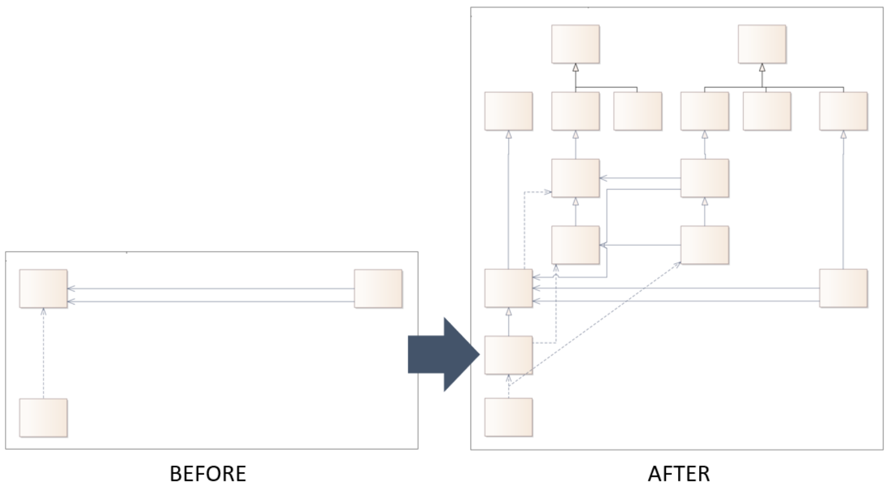

Figure 22 – Data item transformation

During the grounding process, these levels are associated with data items. This enables the evolving data items to be marked with a digitalisation level, as shown in Figure 23.

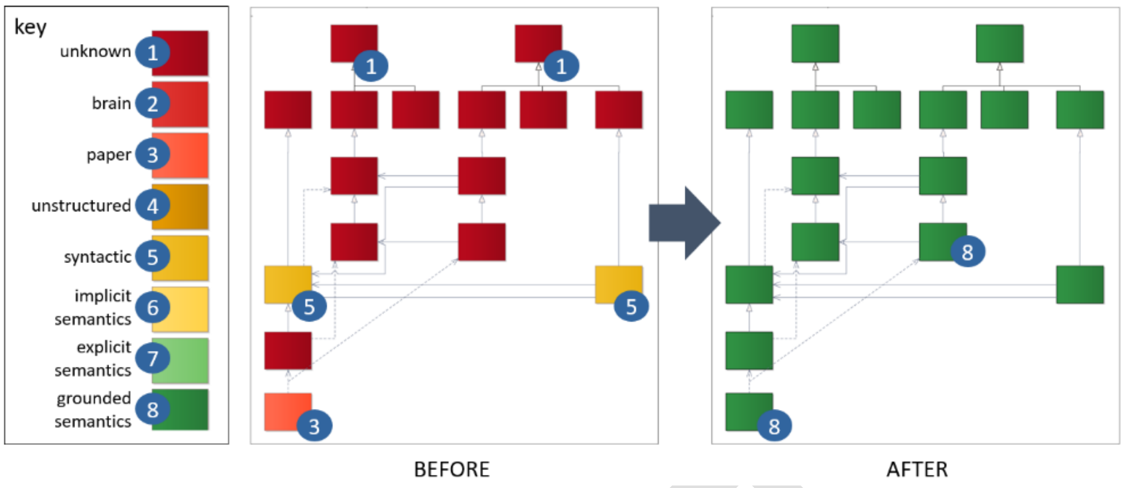

Figure 23 – Assessing the data item digitalisation level

A system of any size will be composed of a large number of data items – and these can be at a variety of digitalisation levels. The aim of the grounding process is to eventually evolve all the data items to the grounded level – as shown in Figure 24. However, what is not shown here, is that there will typically be a number of intermediary stages.

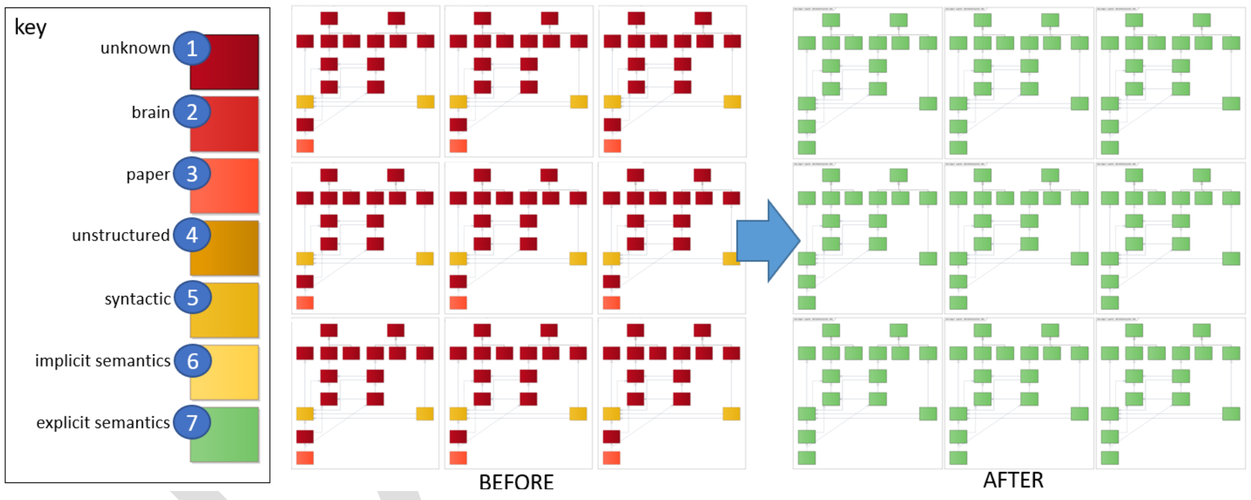

Figure 24 – Digitally transforming a whole system

Appendix F. Glossary

This glossary briefly explains some of the specific terms raised in this report.

Term	Description
FDM Seed	The core of the FMD's TLO. In this case, the categories of the top-level ontology.
ontological pattern (in an ontology)	a recurring set of relations between object with a similar structure in an ontology
ontological commitment (of a dataset)	the objects whose existence the dataset commits to
top-level ontology	the general objects in an ontology that one would expect to find across most if not all domain ontologies (in the case of the NDT's IMF, it is composed of the top-level categories and their organising objects)
minimal foundation (for a thin slice)	a minimal, basis used across thin slices as a common foundation
(top-level) categories	categories are general kinds that exclusively and exhaustively divide the entities committed to by an ontology (top-level ontologies will typically have a system of categories at their top level – hence these are also called the top-level categories)
(top-level ontology) grounding process	the process (in a top-level ontological approach) which grounds a dataset in the top-level ontology

References

Abrial, J.-R. (1974). *Data semantics*. Université scientifique et médicale.

Ackrill, J. L. (1963). "*Aristotle's Categories and De Interpretatione (translation with notes)*". Oxford: Clarendon Press, 4, 1.

Amabile, T. M. & Kramer, S. J. (2011). "*The Power Of Small Wins*". Harvard Business Review, 89(5), 70–80.

Ambady, N. & Rosenthal, R. (1992). "*Thin slices of expressive behavior as predictors of interpersonal consequences: A meta-analysis.*". Psychological Bulletin, 111(2), 256.

Bahrs, P. (2014). *Shifting Left - Approach and Practices*. https://www.slideshare.net/Urbancode/shift-left

Baumer, M. R. (1993). "*Chasing Aristotle's categories down the tree of grammar*". Journal of Philosophical Research, 18, 341–449. https://doi.org/10.5840/jpr_1993_11

Dutilh Novaes, C. (2015). "*The formal and the formalized: The cases of syllogistic and supposition theory*". Kriterion: Revista de Filosofia, 56, 253–270.

Fine, K. (2017). "*Naive metaphysics*". Philosophical Issues, 27(1), 98–113.

Firesmith, D. (2015). "*Four types of shift left testing*". Podcast, Software Engineering Institute Website, September.

Florio, S. & Linnebo, Ø. (forthcoming). *Core Constructional Ontology: The Foundation for the Top-Level Ontology of the Information Management Framework*.

Fonseca, C. M., Almeida, J. P. A., Guizzardi, G. & Carvalho, V. A. (2021). "*Multi-level conceptual modeling: Theory, language and application*". Data & Knowledge Engineering, 134, 101894.

Hetherington, J. & West, M. (2020). *The pathway towards an Information Management Framework- A "Commons" for Digital Built Britain*. CDBB. https://doi.org/10.17863/CAM.52659

Johnson, R. E. (1998). "*Dynamic object model*". Work in Progress.

Lowe, E. J. (2006). *The four-category ontology: A metaphysical foundation for natural science*. Oxford University Press.

Lowe, E. J. (2012). "*Categorial predication*". Ratio, 25(4), 369–386.

Lowe, E. J. (2013). *Forms of thought: A study in philosophical logic*. Cambridge University Press.

Moltmann, F. (2017). "*Natural language ontology*".

Neurath, O. (1973). "*Anti-spengler*". In Empiricism and sociology (pp. 158–213). Springer. https://doi.org/10.1007/978-94-010-2525-6_6

NIC. (2017). *Data for the public good*. https://nic.org.uk/app/uploads/Data-for-the-Public-Good-NIC-Report.pdf

O'Brien, G., Xiao, G. & Mason, M. (2019). *Digital Transformation Game Plan*. O'Reilly Media, Incorporated.

OMG. (2016). "*OMG Meta Object Facility*".

Partridge, C. (forthcoming). *Top-Level Categories: Categories for the Top-Level Ontology of the Information Management Framework*.

Partridge, C. (1996). *Business objects: re-engineering for re-use*. Butterworth-Heinemann.

Partridge, C., de Cesare, S., Mitchell, A. & Odell, J. (2016). "*Formalization of the classification pattern: survey of classification modeling in information systems engineering*". Software & Systems Modeling, 1–37.

Partridge, C., Mitchell, A., Cook, A., Sullivan, J. & West, M. (2020). *A Survey of Top-Level Ontologies - to inform the ontological choices for a Foundation Data Model.*

Partridge, C., Mitchell, A., da Silva, M., Soto, O. X., West, M., Khan, M. & de Cesare, S. (2020). "*Implicit requirements for ontological multi-level types in the UNICLASS classification*". In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 1–8).

Partridge, C., Mitchell, A. & de Cesare, S. (2019). "*Grounding for an Enterprise Computing Nomenclature Ontology*". In International Conference on Conceptual Modeling (pp. 457–465).

Schooling, J., Burgess, G. & Enzer, M. (2020). *Flourishing Systems: Re-envisioning infrastructure as a platform for human flourishing*. https://doi.org/10.17863/CAM.52270

Smith, B. (2005). "*Against fantology*". In J. Marek and E. M. Reicher (Ed.), Experience and Analysis (pp. 153–70).

Smith, L. (2001). "*Shift-left testing*". Dr. Dobb's Journal, 26(9), 56–ff.

Sommers, F. (1983). "*The logic of natural language*". Revue Philosophique de La France Et de L, 173(3).

Ward, M. (1994). "*Language Oriented Programming*". Science Labs.

West, M. (forthcoming). *Managing Shared Data: An introduction to the Information Management Landscape and the Information Management Framework*.

West, M. (2011). *Developing high quality data models*. Elsevier.

West, M., Cook, A., Leal, D., Mitchell, A., Partridge, C. & Sullivan, J. (2020). *The Approach to Develop the Foundation Data Model for the Information Management Framework*. https://www.cdbb.cam.ac.uk/files/approach_summaryreport_final.pdf

Yoder, J. W., Foote, B., Riehle, D. & Tilman, M. (1998). "*Metadata and Active Object-Models*". In Dept. of Computer Science, Washington University Department of Computer Science.

Yoder, J. W. & Johnson, R. (2002). "*The adaptive object-model architectural style*". In Working Conference on Software Architecture (pp. 3–27).

Acknowledgements

Author:

Chris Partridge

Contributors:

Tom Burgoyne
Al Cooke
Ian Cornwell
Anne Guinard
Pierre Grenon
Liam Mcgee
Andrew Mitchell
Justin Price
Matthew West