The Fantastic Combinations and Permutations of Coordinate Systems' Characterising Options: The Game of Constructional Ontology

Chris Partridge
BORO Solutions Ltd
University of Westminster
0000-0003-2631-1627

Andrew Mitchell
BORO Solutions Ltd
0000-0001-9131-722X

Michael Loneragan
QinetiQ Group PLC
Portsmouth, Hampshire, UK
mjloneragan@qinetiq.com

Hayden Atkinson
CooperVision Ltd
0000-0002-5153-9116

Sergio de Cesare
University of Westminster
0000-0002-2559-0567

Mesbah Khan
OntoLedgy Ltd
University of Westminster
0000-0002-1327-6263

... Conway's latest brainchild, a fantastic solitaire pastime he calls "life". ... The basic idea is to start with a simple configuration of counters (organisms), one to a cell, then observe how it changes as you apply Conway's "genetic laws" for births, deaths, and survivals. ... Conway's genetic laws are delightfully simple. ... It is important to understand that all births and deaths occur simultaneously. Together they constitute a single generation or, as we shall call it, a "move" in the complete "life history" of the initial configuration. ... Because births and deaths occur simultaneously, newborn counters play no role in causing other deaths and births. ... You will find the population constantly undergoing unusual, sometimes beautiful and always unexpected change. ...

— Martin Gardner, The Fantastic Combinations of John Conway's New Solitaire Game "Life", 1970 [1]

Philosophy [nature] is written in that great book whichever is before our eyes -- I mean the universe -- but we cannot understand it if we do not first learn the language and grasp the symbols in which it is written. The book is written in mathematical language, and the symbols are triangles, circles and other geometrical figures, without whose help it is impossible to comprehend a single word of it; without which one wanders in vain through a dark labyrinth.

— Galileo Galilei, Il Saggiatore, 1623

No one shall expel us from the paradise that Cantor has created for us.

— David Hilbert – *Mathematische Annalen* 95, 1926, p. 170

Abstract:

The multi-level modelling community's raison d'être is its vision of the ubiquity and importance of multi-level-types: the ascending levelled hierarchy of types in conceptual models; starting with types of things, then types of these types, then types of these types of types, and so on. The community both promotes this vision and investigates this hierarchy, looking at how it can be accommodated into existing frameworks. In this paper, we consider a specific domain, coordinate systems' characterising options. While we recognise that, unsurprisingly, this domain contains a ubiquity of multi-level-types, our interest is in investigating a new and different approach to understanding them. For this we needed to develop a new framework. We devise one focussing on this case, based upon scaling down to simple compositional algorithms (called constructors) to form a new, radically simpler foundation. From the simple operations of these constructors emerges the scaled up multi-level structures of the domain. We show how the simple operations of simple constructors give rise to compositional connections that shape – and so explain – different complex hierarchies and levels, including the familiar multi-level-types and relatively unknown multi-level-tuples. The framework crystallises these connections as metaphysical grounding relations. We look at how simple differences in the shape and operation of constructors give rise to different varieties of these hierarchies and levels – and the impact this has. We also look at how the constructional approach reveals the differences between foundational constructors and derived constructors built from the foundational constructors – and show that conceptual modeling's generalisation relations are secondary and dependent upon the more foundational instantiation relations. Based upon this, we assemble a constructional foundational ontology using the BORO Foundational Ontology as our starting point. We then use this to reveal and explain the formal levels and hierarchies that underlie the options for characterising coordinate systems.

Keywords— Constructional Ontology; Ontological Space; Ontological Sandpits; BORO Foundational Ontology; Grounding Relations; Multi-Level Options; Geometric Coordinate System Ontology; Multi-Platform-Domain Sensor System.

1 Overview

1.1 Introduction

The multi-level modelling community's raison d'être is its vision of the ubiquity and importance of multi-level-types: the ascending levelled hierarchy of types in conceptual models; starting with types of things, then types of these types, then types of these types of types, and so on. The community both promotes this vision and investigates this hierarchy, looking at how it can be accommodated into existing frameworks. In this paper, we consider a specific domain, coordinate systems' characterising options. While we recognise that, unsurprisingly, this domain contains a ubiquity of multi-level-types, our main interest is in investigating a new and different approach to understanding them. For this we needed to develop a new framework. We devise one focussing on this case, based upon scaling down to simple compositional algorithms (called constructors) to form a new, radically simpler foundation. From the simple operations of these constructors emerges the scaled up multi-level structures of the domain. We show how the simple operations of simple constructors give rise to compositional connections that shape – and so explain – different complex hierarchies and levels, including the familiar multi-level-types and relatively unknown multi-level-tuples. The framework crystallises these connections as metaphysical grounding relations. We look at how simple differences in the shape and operation of constructors give rise to different varieties of these hierarchies and levels – and the impact this has. We also look at how the constructional approach reveals the differences between foundational constructors and derived constructors built from the foundational constructors – and show that conceptual modeling's generalisation relations are secondary and dependent upon the more foundational instantiation relations. Based upon this, we assemble a constructional foundational ontology using the BORO Foundational Ontology [2, 3] as our starting point. We then use this to reveal and explain the formal levels and hierarchies that underlie the options for characterising coordinate systems.

1.2 The paper's themes as two tropes

This paper has a focus on examining the case of coordinate systems' characterising options in the light of a BORO constructional ontology. The examination has two main underlying themes which we would like to make explicit. These themes are based upon two tropes; *ubiquity revealed* and *radically simplify by scaling down to scale up* – as we explain below.

1.2.1 Ubiquity revealed

There is a common trope for scientific and engineering advances, which we call *ubiquity revealed*. In this narrative, at the start people work with data but see no pattern, they then learn to recognise the pattern and as a result see it everywhere in the data. The community goes from a state of not being able to see the pattern anywhere, to seeing it everywhere. The surprise here is both that knowledge is hiding in plain sight and that once one learns how to look, it is not only really obvious but everywhere. A reasonably well-known example is complex systems and its many features. Let's pick one; say, scaling/power laws. Once scientists developed an understanding of complex systems and scaling laws, these laws were revealed as ubiquitous in complex systems, visible everywhere.

The multi-level modelling (MLM) community's vision has this *ubiquity revealed* narrative; once one learns to recognise multi-level-types in information systems, it becomes apparent that they are ubiquitous. Part of the work of the community is helping the wider information systems community learn to see these patterns. Another part of the work is examining the phenomena; aiming to explain what these multi-levels are (often the answer is ontological) and the implications of accommodating this pattern with other structures in the scheme of things – for example, how attributes have a potency reflecting their mode of association with their multi-level-types. Our paper will illustrate the MLM narrative for a specific case, coordinate systems' characterising options for sensor data, showing how a close look at these exposes the ubiquitous multi-level-type pattern.

1.2.2 Radically simplify by scaling down to scale up

There is another common trope associated with scientific and engineering advances, which we call *radically simplify by scaling down to scale up* – abbreviated to *scaling down.* In this reductionist narrative, at the start a community is familiar with a complex, often ubiquitous, pattern, but cannot explain it. They then find a way of explaining this in two steps. They firstly decompose the elements of the pattern into simple components – scaling down. Then they consider the patterns that emerge from the simple interactions of these components – scaling up. And they see how their complex pattern, and often other new patterns emerge. In this way, the components and their interactions provide a new, radically simpler (down-scaled) foundation. The community goes from a state of seeing but not being able to explain ubiquitous complex patterns, to having a simple explanation based upon patterns emerging from simple interactions of a scaled-down foundation. The novelty here is the way the scaling down to a simple foundation dissolves and explains the scaled-up complexity.

There are many examples of this in physics' search for fundamental particles – starting as far back as Ancient Greece, with Democritus' atomism. A modernish scientific example starts with the ubiquitous patterns of periodic trends in Mendeleev's periodic table of elements. Empirically, there were clearly patterns, but no real explanation of how they arose. Also, there were oddities; for example, Mendeleev ordered cobalt and nickel based upon their properties rather than atomic mass to make the trend patterns work. Scientists scaling down, looking inside the atomic elements, developed simple structures that explained how these patterns, including the oddities, emerged. Rutherford and Bohr's work [4] on the model of the atom led to electron configurations. While Henry Moseley's work on atomic number – the positive charges on the nucleus – helped to explain the ordering, including that of cobalt and nickel. Breaking down the atomic elements into sub-atomic components provided a foundation which simply explained the patterns. It also created the possibility for new patterns (new periodic trends) based on the components, such as electronegativity. A more recent example from complex systems is Conway's Game of Life [1]. This very neatly illustrates how very simple algorithms give rise to – and so help to explain – the complex 'lives' of cell automata and so perhaps complex natural life.

In this paper, we tell a *radically simplify by scaling down to scale up* story. Given MLM's revealed ubiquitous multi-level-types pattern, we outline a new, radically simpler (down-scaled) foundation that explains not just how the multi-level-types pattern emerges, but also how the whole underlying formal grounding structure, including other patterns of levels and hierarchies, emerges. Much like in Conway's Game of Life, we start with a judicious choice of simple construction processes. These then give rise to a fully-fledged formal domain containing hierarchies and levels, including multi-level-types. This draws upon research done in philosophy, particularly Kit Fine's work on constructional ontology, composition (part-whole), procedural postulation and formal grounding. (This is quite technical, so we found we needed to devote a significant portion of the paper to giving an exposition sufficiently rich to enable a reasonable understanding.) With this understanding in place, we examine the case of coordinate systems' characterising options to show how this approach can be deployed.

1.3 The structure of the paper

The rest of the paper is divided into six broad parts. The first part aims to give a general context by briefly outlining the overall project that framed this work, describing the specific coordinate system characteristics challenge and how we aim to address it. It then describes the initial project that focuses on this specific challenge. The next two parts of the paper contain the technical analysis. The second part is devoted to outlining the general constructional framework (based upon work by Kit Fine, particularly [5], [6].) The third part uses this framework to assemble a constructional BORO foundational ontology. The next two parts of the paper focus on using the technical analysis to develop a constructional ontology for coordinate systems' characterising options. The fourth part gives a brief overview of the approach to the building of the ontology. The fifth part looks in more detail at the ontology's examples of the multi-level options – both type and tuples. A final summary concludes the paper.

2 The requirement for our case

The need to understand the characterising options for coordinate systems arose from a specific requirement which we outline in this section. It is well-recognised that a major challenge currently facing the deployment of collaborating unmanned vehicles is semantic interoperability [7, 8], and that as this technology develops, the requirements for interoperability are likely to become both more stringent and complex. A common (preferably open) data architecture is seen as key to resolving this [7, 8]. Many of these current vehicles use systems and data structures that are proprietary and have a single platform – single domain heritage. These typically made no use of conceptual models in their development, and so have a lightweight (sometimes, non-existent) conceptual framework. It is a situation with substantial opportunities for improvement [9].

In most vehicles, sensors are the major producers of data, with a significant proportion of this being sensed position data. Sensed positions (typically structured as a triple of coordinates) are relative to a coordinate system. Where there are multiple platform/domains, their sensors will use different local coordinate systems. To be able to integrate this sensor data into a single common picture, the integrating system needs to know the various sensed positions' coordinate systems. For example, the integrating system might receive two sensed position triples from different platforms' sensors with the same coordinate numerals (such as <10, 20, 30>). These would typically use different local coordinate systems and if the integrating system does not know which coordinate system each is relative to, it cannot interpret and integrate them. The '20' coordinate in the first triple might be relative to a

Cartesian coordinate system and so refer to a distance and the same 'value' in the second triple be relative to a Spherical coordinate system and so refer to an azimuthal angle – or maybe vice versa. In general, if the integrating system does not know enough about the 'owning' coordinate systems, it cannot interpret the position triples and so integrate them.

More generally, what is required is an understanding of which characteristics of the coordinate system need to be known so that the position triple can be interpreted. Unfortunately, little work has been done on determining what a full characterisation would look like. In practice, coordinate systems are often not explicitly defined at all: it is assumed that users of the sensors know enough about what their coordinate system is. Where coordinate systems are defined, the characterisation is partial and pragmatically ad hoc.

2.1 The project

We are working on a project that is assessing a radical approach to developing a suitable conceptual architecture for articulating the requirements for a full characterisation. The aim is to uncover the underlying conceptual foundations and reveal a clear fundamental picture. And, in so doing, to strip away any pre-conceptions remaining from the single platform/domain heritage. To uncover its conceptual foundation, we are taking a close technical look at the geometric foundations of the world described by the sensor position data.

The project showcases an approach to building a foundational conceptual model that should be capable of resolving the semantic integration problems that multi-platform/domain sensor systems are currently facing. The project's prime analytic tool is a constructional ontology based upon the BORO Foundational Ontology.

Early work has focused on three simple coordinate systems for sensed positions and is clearly exposing an underlying compositional geometric structure; one where systems are built from a common set of ontological components whose construction processes follow broadly similar stages. In this paper, we focus on one aspect that is interesting from a multi-level modelling perspective. We investigate how to characterise the variety of coordinate systems as a series of sets of options. We focus on how these sets of options are embodied in the ontology by generating option objects at a higher level. These option objects are of two kinds. One is the well-known 'type' multi-levels (associated with Powertypes [10]). The other is a second kind of level-ascending based upon 'tuples', also known as 'relations' – which in turn gives rise to another levelled hierarchy. This second pattern is as far as we can determine new to the MLM community; so, from the perspective of the *scaling down* narrative, they are new patterns. We use the types and tuples hierarchies to characterise two kinds of options: combinations (types) and permutations (tuples/relations). These provide a basis for the fundamental characteristics needed to interpret the sensor position**.**

In the following sections, we look at the context in more detail and then give an overview of the requirements of the project. We then note our insights and show how this motivates our approach.

2.2 Context

Unmanned vehicle collaboration across multiple domains/environments (air, surface, land, underwater and space) is recognised as a difficult engineering problem – Figure 1 shows examples of both single and multiple platform/domain manned and unmanned collaborating vehicles.

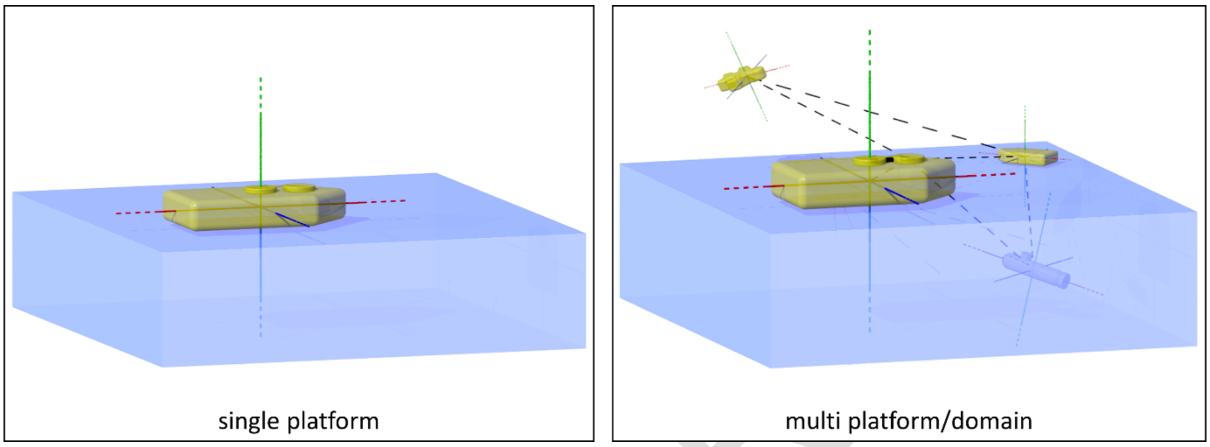


Figure 1 – Single and multi-platform with deictic axes

One challenge is the semantic integration of the sensing data into a single common picture – sometimes known as ‘ground truth’. A common data architecture with agreed data structures and APIs would simplify the challenge at the syntactic level. But this needs to be supported by a common semantic model to ensure shared semantics. The scope of such a model extends beyond the APIs, as their semantic integrity depends upon the systems behind them respecting (and so understanding) it.

The unmanned vehicle sensors process the raw data and pass this on to other, typically centralised, systems for further processing. The level of local onboard processing varies depending upon various factors; for example, low bandwidth restrictions might lead to a preference for onboard over centralised remote processing. The sensors work on a local basis of own position and measure other positions relative to themselves – they may further process these measurements before reporting or directly report a sensed position relative to themselves. Directly reporting the local positions may be preferred as this allows a centralised, consistent calculation of errors.

At the core of these reports is a sensed position recorded using coordinates. The data format of these coordinates is apparently simple and easy to specify – a triple of numbers, with a timestamp. However, it has emerged that it is more of a challenge to find a common data (and semantic) format to characterise all the coordinate systems to which these coordinates can (or could) belong. In large part, this is because the common format will need to be able to accommodate significantly more variety and complexity than the current single platform/domain systems – and include enough detail to make coordinate conversions between the systems, or to a common system. The ways in which the systems vary include:

- *Coordinate system.* Unmanned vehicles should be easy to add to (and remove from) the collaborative sensor systems – whatever coordinate system they use. These vehicles are likely to use new types of coordinate systems which will need to be supported. So, some general structure for coordinate systems needs to be developed.
- *Position and orientation.* The unmanned platforms are moving (with both linear and angular velocity) relative to the main platform in all three dimensions – so the position and orientation of their coordinate systems will be both different and varying. So, some general structure for position and orientation needs to be developed.
- *Angle and unit.* There will typically be limited governance over suppliers of the unmanned vehicles, who are likely to use their own configuration for the coordinate systems. For example, they may use different distance units; one using kilometres, the other miles. So, some general structure for distance and angle units as deployed in coordinate systems needs to be developed.
- *Domain-specific simplifications.* Platforms in the sea domain have, in the past, often had more basic requirements than other domains. For example, they have typically used small angle correction, and some even only considering yaw angular movements, ignoring roll and pitch – as these are not so relevant for single platforms in the sea domain. (The papers [11, 12] describe another simplification for position calculation.) More generally, this raises the requirement, in multi-domain systems, for these domain-specific simplifications to be harmonised to avoid error-generating inconsistencies.
- *Direction.* Different platforms and sensors will use different directions within the orientations. For example, the Cartesian z-axis often has a down direction for underwater and aerial platforms and an up direction for surface platforms (in the maritime domain, this can vary from ship to ship). So, some general structure for directions as deployed in coordinate systems needs to be developed.

2.3 The project's aims

The project aims to build a conceptual model that will support the semantic unification requirements of multiple platform/domain systems. More generally, it aims to showcase a general methodology for designing the data architecture of this domain; one that involves a principled, repeatable, auditable, extendable process. Such a process should be able to identify the range of possible coordinate systems characteristics (possibly exposing their foundations) and design a parsimonious and elegant conceptual model for representing them.

This should provide a degree of comfort that the data architecture built from the conceptual model not only accurately covers current requirements but is also relatively future-proofed:

- that the process will identify a reasonably complete range of possible configurations and
- that it will be easily extendable to new coordinate systems.

It should also provide a benchmark for identifying gaps in the existing data architectures.

2.4 Our insights

The following three insights motivated the approach for developing the conceptual framework outlined in this paper:

1. Each characteristic of the coordinate systems can be thought of as an exhaustive set of independent options. For example, the coordinate system's surface configuration type may be Cartesian, cylindrical, or spherical – one of these options needs to be selected. Generally, the sets of independent options seem to come in two varieties (kinds), combinations and permutations. These correspond, respectively, to ways the system can be and to ways of organising the system.
2. Currently, there is no obvious parsimonious and elegant framework for organising these characteristics waiting to be plucked off the shelf. Standards, such as OMG's [13] and ISO's [14], do not (upon inspection) provide the right kind of help. Neither does theoretical work such as [15]. Though, of course, all of these provide useful input. In some ways, this is a surprising situation, as Euclidian coordinate geometry has been researched extensively for millennia. In other ways, it is not so surprising, as the motivation for this research has not been to unearth the characteristics that should drive a conceptual model to support a data architecture.
3. The coordinate system characteristics that drive the conceptual model are grounded in the system's geometry and that an understanding of these characteristics will emerge from a clear picture of its foundational geometrical features. (As a side note, there is a revived interest in geometry as a mathematical foundation for space and time – see, for example [16] – as well as one in the foundations of Euclid's original geometric work – see, for example, [17].

2.5 Our approach

We decided to start with an ontological conceptual model which would give us a technology agnostic picture. Given the importance of exposing the geometric foundations, we recognised the need to be geometry friendly. We chose a foundational ontology that is extensional and four-dimensional, the BORO Foundational Ontology [3], and are deploying it using a constructional approach [2]. We expected this to not only expose the foundations of the range of possible coordinate systems characteristics but also provide a workspace for exploring the relative parsimony and elegance of different conceptual structures. We also adopted the goal of understanding what the ontology of the coordinate system options is; to enable us to use this to design the data architecture.

As a first stage, we started an initial project for the limited set of the three simplest local coordinate systems; Cartesian (sometimes called rectangular – though from our perspective planar would be more accurate), spherical and cylindrical. We also assumed that we could simplify the geometry to Euclidian affine space-time. We build upon earlier work we have done with coordinate systems [18, 19]. This initial project is under way, and this paper is based upon early results.

3 The general constructional framework

This section is a technical overview devoted to explaining the general constructional framework we have devised for this case. It is divided into six subsections. The first provides a general background. The next two provide the general background framework; the general ontological framework and the general composition framework. The final three sections build an application framework upon this.

3.1 Background

This framework is motivated by a Neo-Aristotelian view of ontology and based upon a specific constructional ontology developed by Kit Fine.

3.1.1 Philosophical motivation

There have been developments in ontology in the last few decades often grouped together under the label ‘Neo-Aristotelian’. This typically takes issue with the flat Quinean view of ontology [20] reflected in Jonathon Lowe’s description in *The Oxford Companion to Philosophy* of ontology as “the set of things whose existence is acknowledged by a particular theory or system of thought.” Instead, it proposes that reality has an underlying metaphysical structure [21, p. 354] – and that this structure is more than a set, more than a sorted list of categories, rather that it is ordered by some sort of metaphysical grounding – these different structures are illustrated in Figure 2. In this paper, we build an ordered ontology – the constructional BORO ontology – and show how it explains the structure of our coordinate systems example.

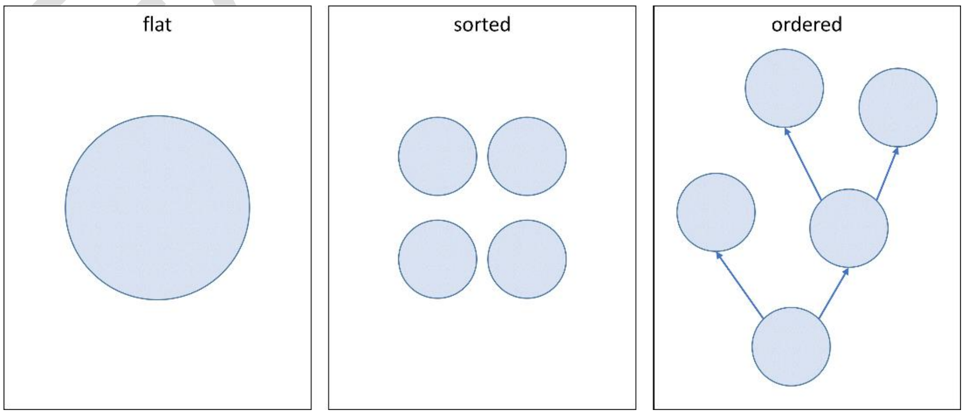


Figure 2 – Types of structure (from [21, p. 355])

3.1.2 *Fine’s constructional ontology*

Constructional ontology has a history that can be traced back through Goodman to Carnap [22]. The particular algebraic constructional ontological framework used here is based upon that outlined in Fine’s [5] and further developed in [6] as a general compositional framework.

Fine makes plain his focus [6, p. 560] is on “the ‘pure’ theory of part whole rather than its application to our actual ontology.” Our focus here is on understanding an actual case – coordinate systems – through the application of an actual ontology. Fine’s pure theory provides us with the starting point for the framework for our endeavour.

Fine sees the specific advantage of what he calls the ‘operational’ nature of his framework is that it naturally leads one to consider the nature of the metaphysical structures. As an example, he comments on levels (a central topic for MLM): “there is an intuitive distinction between wholes which are like sets in being hierarchically organized and those which are like sums in being 'flat', or without an internal division into levels. The distinction, under the operational approach, can be seen to turn on whether repeated applications of the operation are capable of yielding something new.” [6, p. 566]. Elsewhere [6], he talks about its power and beauty; its ability to provide a single and elegant account of a variety of structures. This makes it an ideal tool for the task at hand. For investigating the metaphysical structure of possible ontologies and, in so doing, investigating the ontological content of multi-level-types while developing a new foundation that explains their ontological structure.

In this paper, we build upon Fine’s work to develop a framework for building constructional ontologies – and examine and investigate their constituent constructional components in sandpits. We use this to develop the constructional BORO ontology to help us analyse the example of coordinate systems.

3.2 *Finean general ontological framework*

Fine’s general framework has what he calls two theories; a core theory about what individual constructional ontologies are and an extended theory for multiple ontologies in an ontological space.

3.2.1 *Finean core theory*

For Fine an ontology is constructional if some of the objects of the ontology are accepted (that is, included within the ontology) on the grounds that they are constructed from other objects within the ontology. It is their status as constructs which earns their admission into the ontology. For him the paradigm of a constructional ontology is the cumulative hierarchy of sets [23]. Sets are admitted into the ontology on the grounds that they are constructed from their members. (From now on we will feel free to drop the qualification ‘constructional’ from ontology – as all ontologies discussed here will be constructional in this Finean sense. We will also feel free to deviate slightly from Fine’s terminology and structure where this suits our exposition here.)

Fine calls this constructional approach ‘operationalism’ [6]. He sees it as an example of a specific kind of operationalism which he calls ‘proceduralism’ or ‘procedural postulationism’ [24], [25, pp. 36, 56, 100], where the existence of an object is postulated according to construction rules. Fine explicitly compares these rules to a computer program [24, pp. 90–1] for going from one state of a domain to another and suggests links to dynamic programming logic [26]. He describes the construction process as creative or expansive – as expanding the ontology [24, p. 103] – a point we return to later. He also introduces the metaphor of a genie that automatically executes the procedures.

This metaphor of a genie helps us to visualise construction operations, but we need to be careful to visualise the genie’s work in a platonic, ideal way. We should not think of it happening in space or time. If we do, it would (for example) seem possible for the genie to execute the same operation, creating exactly the same object, at different places or times. This seems a clear case of two executions, where presumably the first execution creates the object and the second merely reconstructs it.

To avoid thinking like this, it is better to idealise the metaphorical genie’s work. One can assume the genie surveys everything that exists and determines all possible operations and executes them simultaneously. This may generate new objects, which may open up the possibility for new executions. The genie will again survey everything that exists and determines all possible operations and execute them simultaneously. This sequence of simultaneously operations is a feature of constructional approaches. Conway’s Game of “Life” which is constructional (though not an ontology) works in a similar way – (Gardner [1]) – quoted at the start of this paper – says “It is important to understand that all births and deaths occur simultaneously” and “each simultaneous execution is called a generation or “move””. In Boolos’ constructional cumulative set theory [23], they are called ‘stages’.

Fine describes how using construction operations shape the architecture of the ontology. They make objects – the constructs. The acceptance of a construct into an ontology requires firstly that, if required, there are constructees (the objects, if any, from which the construct is constructed) and secondly the constructor (the means by which the construct is constructed from the constructees). Of course, constructs, once constructed, can be constructees in later construction operations.

Not all the objects in an ontology need to be constructs, some can be just accepted into the ontology. These are given objects (givens), which (along with the constructors) seed the ontology. From these all the constructs are constructed. An ontology does not need to have givens. For example, in pure cumulative set theory, there are none; the empty set is built from zero input. Similarly, an ontology does not need to have any constructors, all its objects could be givens – though for us this would be an uninteresting limit case – and not constructional under Fine’s definition.

This gives us a framework where objects are accepted into the ontology on one of two grounds, that they are

1. given objects that are just accepted
2. constructs or constructed objects which have been constructed from objects already in the ontology; where a constructor is applied to the constructees to produce constructs.

This leads to a four domain architecture for constructional ontology universes (see Table 1 and Figure 3).

Table 1 – Ontology domains in the ontology universe (see [5])

Acronym	Domain	Names for Members	Descriptions
U	ontology universe	items	contains the ontology domains
O	object domain	objects	all objects in the ontology
CR	constructor domain	constructors	
G	given domain	givens or given objects	a sub-domain of the object domain
CD	constructed domain	constructs or constructed objects	a sub-domain of the object domain

Figure 3 shows the domain composition of the universe more directly. The tree view shows how the given and constructed domains combine to form the object domain. It also shows how the object and constructor domains combine to form the ontology universe. The iconic (Euler) view shows more concretely how the universe is composed.

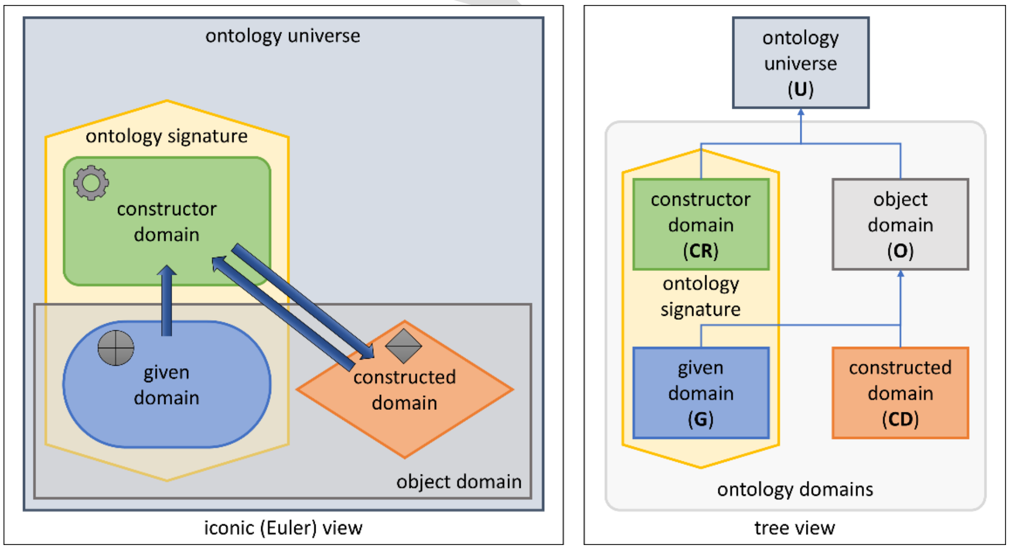


Figure 3 – Two views of the ontology universe's domains

The core theory uses ontological principles to show that one can generate all the constructed objects (the constructed domain **CD**) from the given (**G**) and the constructor domains (**CR**). As Fine notes, this means we do not need the object domain, **O**, to characterise an ontology, we can use just the domain couple <**G**, **CR**> as seeding. We call this the ontology signature – as shown by the hexagonal icon in Figure 3. We will use this signature to characterise ontologies in the rest of the paper.

Given the signature, **CD** can then be generated from **G** and **CR** – and combined with **G** to give **O**. Though the order of analysis may well be the opposite; where one starts with the objects and works out what the constructors are and so the given objects. Furthermore, the signature couple <**G**, **CR**> has more structure than just plain **O**, as it picks out the given objects and the constructors – which gives the rules for how the other (constructed) objects are constructed.

This generation of **CD** relies upon an exhaustive application of the constructors; where anything that can be generated is generated. In our approach, we find it useful to make this process explicit as it helps us to see the structure of **CD**. We call this the ontology's ONTOGENESIS and adopt the triple <**G**, **CR**, **ONTOGENESIS**> as the extended ontology signature, where required. We will be looking at examples of **ONTOGENESIS** later.

3.2.2 Finean extended theory

Fine's extended theory deals with how ontologies fit into an ontological space; where this is a nonempty collection of ontologies that conforms to certain principles. It describes how, given ontologies in a space, similar ontologies with permutations of given and constructor domains also exist in the space. We extend this to permutations of the individual constructors in the domain.

For us, this provides a framework for an incremental sandbox approach for explaining and understanding an ontology. Under this approach, one might start with an idea for a target ontology; characterised by its signature with the given and constructor domains. One can then build an ontological space that has ontology universes for each possible permutation of givens and constructors – as one knows their signatures. One would typically start with the simplest permutations, ontology universes that contain just the given domain or just a single constructor and examine these – looking at the results of applying ONTOGENESIS. One can then pick and examine richer and richer combinations, seeing how these lead to richer structures, until one arrives at the target ontology.

Let's make this more concrete with a very simple example. Consider a target ontology with the signature <(**g**), (**cr**)>; where the given domain has a single object **g** and the constructor domain a single constructor, **cr**. The possible permutations are:

NULL = <**Ø, Ø**>, <(**g**), **Ø**>, <**Ø, (cr)**>, TARGET = <(**g**), (**cr**)>

We can construct an ontological space, where each of these permutations is the signature for an ontological universe. The space is shown in in Figure 4, along with arrows showing how the permutations build up to the target. Working through the ontologies, starting from the simplest combinations allows one to see what an individual contributes to an ontology and how it interacts with other items usually giving an insight as to how it contributes structure to the target ontology.

Below, when we use an ontological space to build the BORO Constructional Ontology, there will be specific examples of this ability to provide insight.

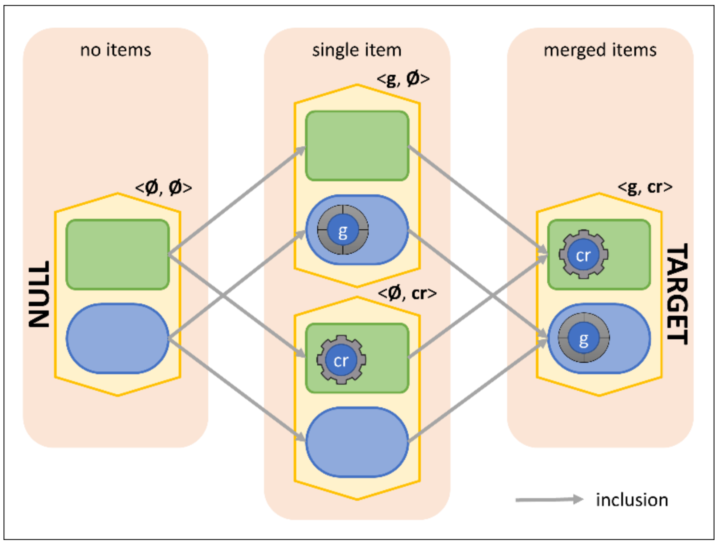


Figure 4 – Example ontological space

3.3 *Finean general composition framework*

Fine [6] sketches a general unified framework for composition; the ways in which one object can be a ‘component’ of another, within his general ontological framework.

3.3.1 *A liberal notion of composition*

Fine’s framework is distinctive in several ways (all of which suit our current purpose). The composing or part relation is usually restricted to mereological parts – where, for example, my hand is a part (component) of my arm. Fine proposes a very liberal notion of composition that includes many types of component, where the traditional mereological relations are just one type. So, for example, it would include a member of a set being part of the set in a similar way to my hand being part of my arm – though these would be different kinds of part. Fine [6] makes a strong case for this position. We take it as a starting point in this paper.

(Terminologically, Fine chose to use the terms ‘part-whole’ and ‘composition’ to characterise the family of relations, to reinforce their unity. For our purposes, it is more convenient to use ‘composition’ as the term for the family and (following standard practice) reserve the terms ‘whole-part’, and its equivalent alternative ‘part-whole’, for the mereological relation. This is just a terminological matter and in no way undermines Fine’s general unification thesis.)

The formulation of the framework follows the general framework, described above, of givens and constructors. Central to this is identifying the variety of constructors that account for the different ways things can be composed.

3.3.2 *Compositional principles*

Fine formulates his framework in terms of compositional principles. He first divides these broadly into formal and material principles. Our interest here is in the formal principles. He further divides formal principles into those that deal with conditions of application and those that provide identity conditions.

Fine develops a very simple way of characterising the formal identity principles for compositional identity based upon a notion of regular identity conditions (the reader can find the details in [6]). The result is the four CLAP principles – so-called because of their initials –described in Table 2.

Table 2 – CLAP (formal identity) principles (see [6])

C	Collapse	$\sum(x) = x$	If Collapse holds then any composite composed of a single component is identical to it.
L	Levelling	$\sum(\ldots, \sum(x, y, z, \ldots), \ldots, \sum(u, v, w, \ldots), \ldots) = \sum(\ldots, x, y, z, \ldots, \ldots, u, v, w, \ldots, \ldots)$	If Levelling holds then when the components of composite have components, these components' components are also components of the whole.
A	Absorption	$\sum(\ldots, x, x, \ldots, \ldots, y, y, \ldots, \ldots,) = \sum(\ldots, x, \ldots, y, \ldots)$	If Absorption holds then the repetition of components is irrelevant to the identity of the composite.
P	Permutation	$\sum(x, y, z, \ldots) = \sum(y, z, x, \ldots)$ (and similarly for all other permutations)	If Permutation holds then the order of the components is irrelevant to the identity of the composite.

A constructor's formal identity can be characterised by whether these principles are adopted or rejected – one can summarise this into a CLAP profile, with a mnemonic where the appropriate letter is struck through when the principle is rejected.

There are two conflicting senses of the term 'level' that make Fine's choice of the name 'Levelling' less than ideal for the purposes of this paper. There is the sense of making flat, (into a single) level that motivates Fine's use and then there is the almost opposite sense MLM uses of not being flat and having multiple different levels (also used by Fine [6, p. 566] – "in being 'flat', or without an internal division into levels"). We have stuck with Fine's choice – partly to preserve the easy to remember CLAP acronym (CFAP does not have the same ring). And we have tried to make clear in the paper which sense is being used – restricting the present participle 'Levelling' to the Finean sense of making one thing level and the noun 'level' for MLM's sense of arranged in levels.

The other formal element in the framework is the conditions of application – typically what objects can be applied to the constructor. One example, often mentioned in Fine, is whether an empty application is allowed – in other words, no constructees are supplied to the constructor. In this case, it constructs a null object for that type of constructor. There are cases of this in mathematics; for example, the null set for sets and the null thing for things. Ontologies are simpler without null objects. So, for simplicity, in the example ontologies (and later the BORO constructional ontology) in the rest of the paper a null application will not be allowed unless explicitly specified.

The formal identity principles may appear arcane, but they have simple, intuitive, constructive tests. For Levelling, one can consider what happens when there are two applications of the constructor to a simple object. And whether the initial object is a component of the object created in the second application. For example, if one starts with *Socrates* and creates a set from it then one gets {*Socrates*}. If one then creates a set from this, one gets {{*Socrates*}}. And *Socrates* is not a part/component/member of {{*Socrates*}} – so the Levelling principle has not been adopted.

Permutation is about whether order is considered. We will be talking about order quite a bit going forward: to make it clear when we are representing order, we use the less-than sign symbol '<'; where (*a*<*b*) means (*a then b*) in that order.

For Permutation, a simple test is to consider whether changing the order of a couple of objects gives rise to a different object. So, whether the constructor when applied with (*a*<*b*) — constructs a different object than when applied with (*b*<*a*). It does for STRING-BUILDER but not for SET-BUILDER – so the first adopts Permutation and the latter does not. Similarly, for the other two principles.

The formal conditions of application also mostly have simple intuitive constructive tests. In the null object example above, one just needs to ask whether one can make an empty application to the constructor.

3.4 Developing the CLAP formal identity principles

3.4.1 Four familiar kinds

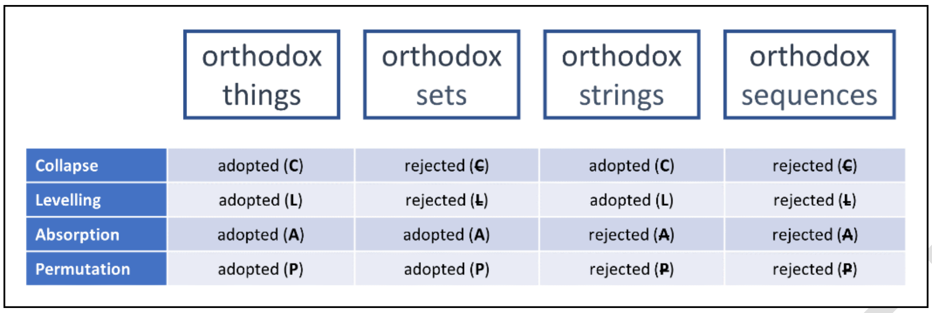
To see how the CLAP principles work and how constructors fit a CLAP profile, consider the four familiar (to mathematicians and computer scientists) cases of composition – described in Table 3.

Table 3 – Four familiar cases of composition (see [6])

Kind	Form of composition
Things	The form of composition is fusion. A typical example is mereological fusion – where the fusion of the parts is the whole. Note: the mereological fusion of a single part is identical to the part itself.
Sets	The form of composition is collection. A plurality of objects is collected into a single new object. Note: the set with a single member – so, for example, the set composed of *Socrates* – {*Socrates*} – is not identical to the member.
Strings	The form of composition is simple concatenation. Concatenating two strings, say *xy* and *uv*, is the same as concatenating some combination of their components; so *x*, *y*, *u*, and *v* or *xyu* and *v* or … and so on. Similar to list data structures.
Sequences	The form of composition is sequence-building. Sequencing two sequences (*xy*) and (*uv*) to obtain ((*xy*)(*uv*)) is to be distinguished from sequencing *x*, *y*, *u*, and *v* to obtain (*xyuv*) (in contrast to the case of strings). Similar to list of lists data structures.

3.4.2 *Four CLAP Profiles for the familiar cases*

These familiar cases can be characterised by CLAP profiles, which we call orthodox – as shown in Figure 5. This captures their formal identity principles, but not the formal conditions of application. So, not, for example, whether they accept null applications.



	orthodox things	orthodox sets	orthodox strings	orthodox sequences
Collapse	adopted (**C**)	rejected (**~~C~~**)	adopted (**C**)	rejected (**~~C~~**)
Levelling	adopted (**L**)	rejected (**~~L~~**)	adopted (**L**)	rejected (**~~L~~**)
Absorption	adopted (**A**)	adopted (**A**)	rejected (**~~A~~**)	rejected (**~~A~~**)
Permutation	adopted (**P**)	adopted (**P**)	rejected (**~~P~~**)	rejected (**~~P~~**)

Figure 5 – CLAP profiles of orthodox versions of the familiar cases

3.4.3 *Levelling and Permutation divide up the cases*

The four CLAP principles have fourteen potential valid combinations (of the sixteen possible combinations, **CLA~~P~~** and **~~C~~LAP** should not be allowed as they are invalid – they lead to cycles [6]). The four profiles in Figure 5 give four formally orthodox cases – which seem to construct different kinds of objects.

How do the remaining ten combinations work? Do they, for example, construct the same or different kinds of object? Firstly, let's establish that two constructors with different CLAP profiles can construct identical objects. Intuitively the answer seems to be that it is possible. The following example confirms this intuition.

Consider these examples of different CLAP profiles listed in Table 4 that appear to be set variants. There is a partial taxonomy for them shown in Figure 6.

Table 4 – Example variants of set

Kind	Profile	Name	Description
Set	~~CL~~AP	formally orthodox sets	defined by the CLAP profile
Set	C~~L~~AP	Quinean non-multi-sets	like orthodox sets but with the singleton identical to its sole component
Set	~~CLA~~P	non-Quinean multi-sets	like orthodox sets but allowing multiple occurrences of the same component
Set	C~~LA~~P	Quinean multi-sets	like both Quinean sets and multi-sets

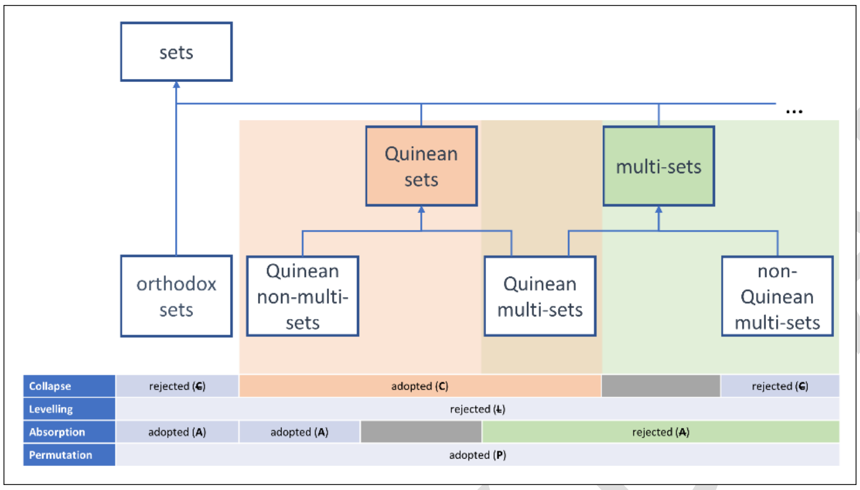


Figure 6 – Partial taxonomy of set variants

Table 5 shows that for a standard input of (a, b, c) there is a standard output of {a, b, c}. It appears to make sense that the different constructors are creating the same set {a, b, c}. The edge cases, where the input to Quinean sets is singular or the input to multi-sets includes repetition, end up having objects generated by one constructor but not the other. But they do not appear to generate counterexamples or inconsistency.

Table 5 – Example inputs and outputs

Profile	Name	Input	Output
~~CL~~AP	formally orthodox sets	(a, b, c)	{a, b, c}
C~~L~~AP	Quinean non-multi-sets	(a, b, c)	{a, b, c}
C~~LA~~P	Quinean multi-sets	(a, b, c)	{a, b, c}
~~CLA~~P	Non-Quinean multi-sets	(a, b, c)	{a, b, c}

Alternatively, one could argue that the constructor was part of the identity of the constructed object. In this case, ~~CL~~AP: {a, b, c} and C~~L~~AP: {a, b, c} (where the CLAP prefix is a label for the generating constructor) would be different objects – despite them being effectively indiscernable in terms of kind and members. This would greatly inflate the framework with redundant objects, so we discount it here.

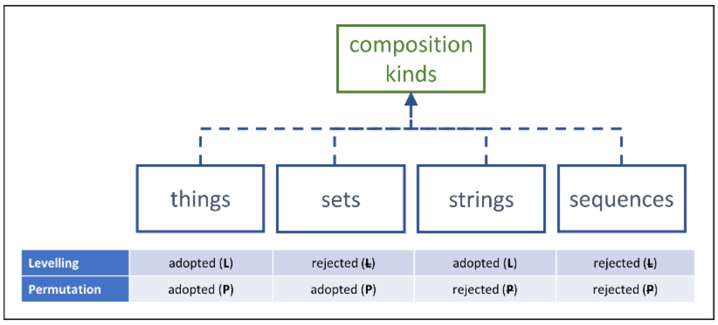
It turns out that some principles influence what kind of object is constructed – these are kind-characterising. Furthermore, it also turns out that the group of kind-characterising principles are sufficient to uniquely characterise the kinds. So, where two constructors similarly adopt or reject this group of principles, they build the same kind of object – where they are not similar, they generate different kinds of objects. Two of the four principles, Levelling and Permutation, form the kind-characterising group. Collapse and Absorption (and the conditions of application) are not kind-characterising.

We can explain this by looking at when two constructors with different CLAP profiles construct identical objects. Assume that they do. Then one would expect that applying identical components to the two constructors would generate identical objects (as in the set example above). This creates a one-to-one mapping between the identical objects. As the example above shows, there can be edge cases where there is no identity and so no mapping.

If two constructors have different Levelling or Permutation principles, then it is not possible to have an identity mapping. Consider Permutation first. We start with two components *a* and *b* as givens and two constructors one adopting, one rejecting Permutation. Assume that there is an identity mapping. Take the Permutation-adopting constructor; at the first stage, the genie applies (*a*<*b*) and (*b*<*a*) constructing two different objects – *ab* and *ba*. If a constructor rejects Permutation, then the genie can only apply (*a*, *b*) – as it does not recognise order – constructing a single object – {*a*, *b*}. Either this single object is identical to both the Permutation-constructed objects or neither. It cannot be identical to both, as this would imply the two Permutation-constructed objects are identical – given identity is transitive. So, it is identical to neither, in other words, there is no identity mapping. Similar arguments apply to Levelling.

Thus, the four combinations of Levelling and Permutation – shown in Figure 7 – characterise four different kinds of objects. Fine sees these four kinds as just a small sample of the possible kinds – for our purposes here we restrict ourselves to them.

		Levelling	
		L	Ⱡ
Permutation	P	things	sets
	Ᵽ	strings	sequences



Levelling	adopted (L)	rejected (Ⱡ)	adopted (L)	rejected (Ⱡ)
Permutation	adopted (P)	adopted (P)	rejected (Ᵽ)	rejected (Ᵽ)

Figure 7 – Kind characterising principles (based upon [6, p. 574])

Furthermore, the degree or similarity is dictated by whether they share principles. Here the rows in the table in Figure 7 (things and sets or strings and sequences) and columns (things and strings or sets and sequences) indicate similarity (as they both either adopt or reject Levelling or Permutation). Whereas the two diagonals (things and sequences or strings and sets) take opposing positions on these principles.

With this in place, there is a framework to let the constructors determine the (sort) kind of the objects they construct. There is not quite enough structure yet, to let them determine the kinds for the whole ontology. The kinds of the given objects cannot, in principle, be given by the constructors – as they are not generated by constructors – though they might be re-constructed. In this case, the simplest solution is to mandate that the kind of each given must be stipulated.

3.4.4 *Stipulated conditions of application*

Given these four kinds, one can ask whether there are conditions of application based upon them. For three of the kinds, it is customary for there to be no constraints; to allow any kind to be included in the application. However, for one kind, *thing*, there is a constraint. Only objects of the kind *thing* can be applied to thing constructors. These conditions are listed in Table 6.

Table 6 – Customary stipulated conditions of application

Kind	Component Kind	Composite Kind
Thing	Thing	Thing
Set	Any	Set
String	Any	String
Sequence	Any	Sequence

3.4.5 *Syntactic expansion and exhaustion*

As discussed earlier, these applications are executed as constructions in simultaneous stages. The metaphorical genie executes all possible constructions and no more, at each stage as part of the ONTOGENESIS process. As is traditional (see [23, p. 221]), we allow a finite, infinite or even transfinite number of stages. The genie is powerful enough to execute these stages if required.

What counts as a possible construction? At any stage, the genie could take any combination of objects in the ontology and apply them. However, this leads to a situation where if there is a possible combination at any stage, then it could be a possible combination at all later stages. So, the process is inexhaustible, even when the process arrives at a point where the same combinations are being executed every time. For deterministic constructions, a more economical strategy is to only consider combinations possible in the stage they first appear (see [23, pp. 221–2] "… sets are formed over and over again … We could continue to say this if we liked; instead we shall say that a set is formed only once, namely, at the earliest stage at which, on our old way of speaking, it would have been said to be formed."). So, the genie only executes each combination of input and constructor once – as this combination always gives the same output, so executing again is pointless.

For example, assume that we have an ontology STRING-EXAMPLE-1: <(*a*, *b*, *c*), (STRING-BUILDER)>. The orthodox string constructor (see Figure 5) rejects absorption so allows repetition. Hence, any given can be repeated any number of times and applied to the constructor. This would lead to very crowded figures and tables. To keep things simple, this string ontology adopts absorption – ignoring repetition – with this profile **CLA~~P~~**.

At the first stage, one of the operations the genie applies is the ordered givens (*a*<*b*<*c*) generating the new string *abc*. At the second stage, no new objects are generated. For example, *a*, *b* and *c* are available for application, but the genie no longer applies (*a*<*b*<*c*) to STRING-BUILDER as it has already executed this operation before. Some objects are reconstructed though, and this is visible in Figure 8.

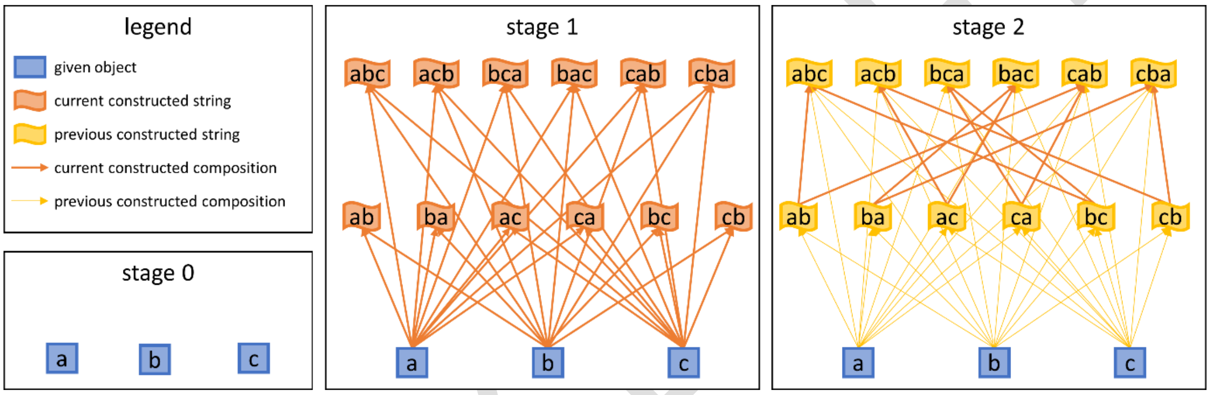

legend
given object
current constructed string
previous constructed string
current constructed composition
previous constructed composition
stage 0
a
b
c
stage 1
abc
acb
bca
bac
cab
cba
ab
ba
ac
ca
bc
cb
a
b
c
stage 2
abc
acb
bca
bac
cab
cba
ab
ba
ac
ca
bc
cb
a
b
c

Figure 8 – STRING-EXAMPLE-1 – visualisation of stages 0 to 2

For non-deterministic constructions, the economical combination is expanded to consider outputs as well as inputs. A possible construction is one where the combination of both the input and the output have not yet been constructed. A constructor is exhausted where, for each of all possible input combinations, all the possible output combinations for a particular input combination have been constructed.

For example, assume that we have an ontology NON-DETERMINISTIC-EXAMPLE: <(*abc*), (RANDOM-STRING-SELECTOR))>, where the constructor returns a random atomic part of the input. At the first stage, the genie applies (*abc*) to RANDOM-STRING-SELECTOR randomly generating the string *b*. At the second stage, the genie again applies (*abc*) to RANDOM-STRING-SELECTOR generating the string *c*. At the third stage, yet again the genie applies (*abc*) to RANDOM-STRING-SELECTOR randomly generating the string *a*. Then the possible combinations of input and output are exhausted – so there are no possible constructions.

With this 'economical' sense of possible in place, the process is complete when all the possible combinations have been executed once. We call this syntactic exhaustion as it only considers the combinations.

3.4.6 *Levelling: single or multiple possible build constructions*

One result of adopting the Levelling principle is that it allows multiple possible constructions for the same object; and rejecting Levelling leads to a single possible construction. Consider these two examples.

Firstly, take the Levelling-adopting STRING-EXAMPLE-1 ontology again. The string *abc* is built (constructed) in multiple ways; by applying ($a<b<c$) or ($ab<c$) or ($a<bc$). In the ontology STRING-EXAMPLE-1, ($a<b<c$) is constructed at both the first and second stage. Clearly, the operation at the first stage will generate the object. The subsequent, multiple second stage constructions merely re-construct the (already generated) object. This allows us to distinguish between two types of construction; *generative*, where new objects are created and merely *compositional*, where no new objects are created. The compositional constructions have a role to play, as they reveal the composition structure. In this case, two composition relations are revealed; that *ab* and *bc* are components of *abc* – as shown in Figure 8. However, note that the overall operations are only partially compositional, as *a* and *c* composing *abc* has already been executed.

It is not always possible to make this distinction. In cases where the multiple constructions are effectively simultaneous, there is no way to order them and say which is first. One could say they are jointly generative.

Now assume that we have an ontology SET-EXAMPLE-1: <(*a*, *b*, *c*), (SET-BUILDER)> with the Levelling-rejecting constructor SET-BUILDER – see Figure 9. At the first stage, one of the operations the genie applies the (unordered) givens (*a*, *b*, *c*) to SET-BUILDER generating the new set {*a*, *b*, *c*}. This is the only way this set can be constructed – and with the 'economical' sense of possible, the only time it is constructed.

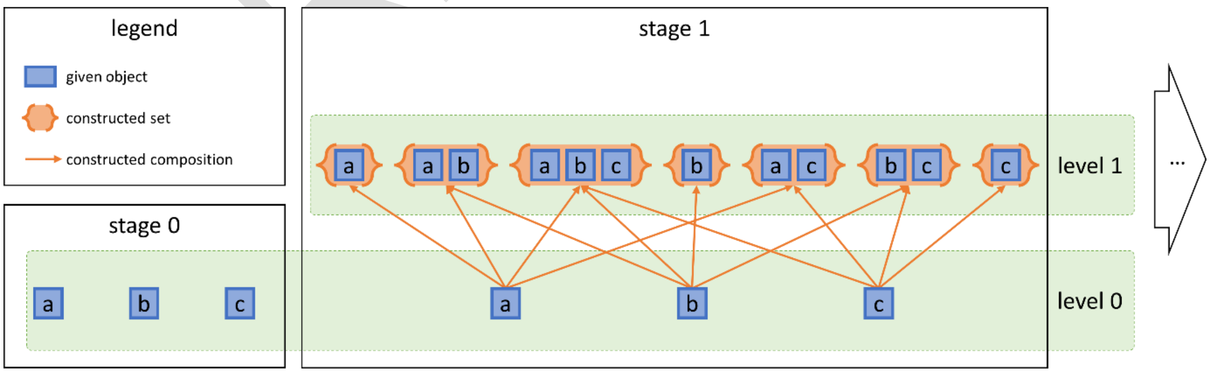


Figure 9 – SET-EXAMPLE-1 – visualisation of stages 0 and 1

3.4.7 *Direction of operation*

Bennett [27] identifies a key choice not captured by the CLAP principles: the metaphysical direction of operation. The operation of a constructor can either start with components and end with their composite (composing) or start with a composite and end with its components (decomposing). In the literature [27], the direction is typically composing. Bennett offers Schaffer [28] as an example of decomposing.

For three of the kinds we are looking at (sets, strings and sequences), the construction of the composite from its components is a formal affair with a natural composing direction of operation. Cantor [29] famously wrote about a set being “a gathering together into a whole of definite, distinct objects” – suggesting a natural direction of gathering together rather than splitting apart. In the constructional approach, this gathering is a construction that postulates the existence of the set built from its members using formal rules: the postulation of strings and sequences similarly involve formal building rules. Also, at least in the simple cases, the constructor is complete – it constructs all objects of these kinds in the same way – no objects of these kinds are introduced as givens.

In the fourth of our kinds, *thing*, there is not a natural direction. Fine [6, pp. 585–6] uses an example to illustrate the need for decomposition. Consider a universe of physical atoms which are physically indivisible but of finite volume, where one can distinguish between the upper and lower parts of the atom (relative to its orientation at a given time). Take the atoms as givens in our ontology. The genie can formally fuse these atoms to construct larger and larger wholes – a formal THING-BUILDER operation. But decomposing an atom into its upper and lower parts is not a formal operation – it depends upon the ability to execute the (material) operation of dividing them. The metaphorical genie needs this material ability to carry out the THING-DIVIDING operation. In this example, THING has both building and decomposing operations, which are individually incomplete, but together complete. This example also indirectly illustrates a concern with THING-BUILDER. As it is possible that things are gunky (infinitely divisible) rather than hunky [30] – then it is possible that in principle they cannot all be built up.

Table 7 summarises the natural directions of operation.

Table 7 – Natural direction of operation

Kind	Natural Direction of Operation for Constructor
Things	no natural direction
Sets	composing (e.g. SET-BUILDER)
Strings	composing (e.g. STRING-BUILDER)
Sequences	composing (e.g. SEQUENCE-BUILDER)

3.4.8 A single mereological maximal atom

For BORO (and Fine [6, p. 585]) the interesting case is along the lines of Schaffer’s priority monism [28], where the given is a single mereological maximal atom – the pluriverse. From this, a thing decomposing constructor, with a material decomposing ability, constructs all the component things.

Fine gives segmentation as an example of material decomposing, which has as input a material thing and a spatio-temporal extension. This constructs parts one by one and keeps the decomposition algorithm deterministic. However, it presumes the separate existence of material things and spatio-temporal extensions, perhaps as givens. BORO is more parsimonious. It adopts super-substantivalism [31], [32] where, among other things, this distinction (between matter and space-time) evaporates. In this situation, a simpler constructor is needed and we adopt the THING-DIVIDER constructor (we use a different name to distinguish it from the more generic THING-DECOMPOSER). For this, the metaphorical genie has the power to divide a spatio-temporal thing into two disjoint parts. Technically, there is no constraint upon the shape of the two parts: for example, they can be as big or small as it is possible given the thing being divided – or as gerrymandered as is possible, so potentially not continuous. As we wish THING-DIVIDER to be Collapse-adopting, we allow a limit case, where the genie division returns just what was applied. This ensures that the THING-DIVIDER constructs improper parts.

In the first stage, the input is the pluriverse, a fusion of all possible things. The genie simultaneously applies this to the THING-DIVIDER constructor until all possible outputs have been constructed – every possible part of the pluriverse has been generated. However, ontogenesis is not complete as all possible wholes-parts have not been constructed. This happens at stage two, were the population after stage one is processed.

An example with a very simple pluriverse will illustrate this. Consider the ontology THING-DIVIDER-EXAMPLE-1: <(*abc*), (THING-DIVIDER)> where the mereological sum of the atoms *a*, *b* and *c* is the given. At the first stage, all the possible parts are generated; at the second stage, all the remaining wholes-parts relations are traversed by the construction, though no new parts are generated – as shown in Table 8.

Table 8 – THING-DIVIDER-EXAMPLE-1 – Stages one and two – possible input-output combinations

Stage	**Input**	**Outputs**	
1	*abc*	*a*	*bc*
1	*abc*	*b*	*ac*
1	*abc*	*c*	*ab*
2	*bc*	*b*	*c*
2	*ac*	*a*	*c*
2	*ab*	*a*	*b*

3.4.9 Operation and composition – alternative bases for ordering

From the perspective of composition, there is a natural ordering – where ‘smaller’ components compose ‘larger’ composites. The introduction of direction (above) gives another natural ordering based upon the direction of operation, which may be different if the ontology contains decomposing constructors. From an ontological perspective, the operational ordering seems a better candidate for (ontological) priority.

3.4.10 Broad extensional-compositional identity

The compositional approach provides an opportunity for a general notion of extensional identity. Two composites of the same kind are identical if they have exactly the same components. There is a sense in which the extension is the components. This has to be all the components, not just sufficient components to recreate the composite. For example, the string *abc* can be deconstructed into *a* and *bc* or *ab* and *c* – and then reconstructed from these components, but neither of the deconstructions are sufficient for identity. For extensional identity one needs the full, exhaustive decomposition: *a*, *b*, *c*, *ab*, *ac* and *bc*. Then composites are identical if their full decompositions are identical.

This notion of full, exhaustive decomposition needs broadening in two ways. Where repetition and order are not involved, full exhaustive decomposition tracks identity. Where either repetition or order are involved, there will typically be different composites with identical decompositions. For example, *aa* and *a* have identical decompositions – *a*. So, where repetition and order are allowed, one adds these to the mix – giving a broader form of extensionality, one sensitive to structure. Identical composites have the same components, the same number of times, in the same order, where these are appropriate.

Null objects, it may seem, are also potential exceptions to basic compositional extensionality. If one takes extension to be components, then as they have no components, surely they have no extension. However, if one allows for zero extensions, then the multiple null objects (including the null set and the null thing) have zero extensions and this gives them identity within their kind – a broad compositional-extensionality criterion of identity.

3.5 Developing the application

Fine makes very clear he is only providing a sketch of the framework and (as noted above) that he is focussing on pure theory rather than application. We are also providing a sketch, but our interest in this paper is in a different area, the application. The pure theory provides a good background framework for us, but we needed to develop this to support application for the specific constructors (and so ontology) we are focusing on. This is a substantial undertaking, and we have pursued the development as far as we needed to for this exercise. There are many opportunities for further development.

3.5.1 What do we apply to a constructor?

Fine talks about applying objects to a constructor; from a computational or algorithmic perspective one would say that these objects are the constructor's input. From an ontological perspective, the obvious question is what these inputs are. One is faced with a kind of circularity, that the constructor is building a structure that is somehow supplied through the input – one is supplying a structure that has not yet been constructed.

Fine [24, p. 93] in discussing the inputs to a SET-BUILDER talks about taking a plurality of pre-existing objects. Pluralities [33] are a good candidate for inputs to SET-BUILDER – Linnebo [34, p. 145] talks about pluralities forming a set. Cantor's [29] comment that a set is "a gathering together into a whole of definite, distinct objects" makes much the same point. Pluralities are normally not regarded as objects but as ways of referring to a multitude (plurality) of individual objects without commiting to referring to the multitude itself as an object – the referring does the work of collecting the multitude.

It makes sense to regard the plurality as extensional – where co-extensional multitudes are regarded as the same [33], [35, p. 22]. If one refers many times to a co-extensional multitude of objects, there is a sense in which one is referring to the same plurality. If one then refers to a different multitude (say, with one less object) then it is a different plurality. In a sense, co-extensionality is doing the work of identity.

There is no standard notation for listing the members of pluralities, probably due to concerns about whether this might unintentionally seem to commit to a plurality object. However, in this paper it will be useful to have a notation, so we use the following. For the simple unordered pluralities we will use ordinary round brackets, comma separators, constants *a*, *b* and variables *x*, *y*, … – for example (*Socrates*, *a*, *b*) or (*x*, *y*). For a variable plurality, we use (*xx*) following the usual *xx* in plural logic. For a variable singleton plurality, we use the variables *x* and *y* – e.g. (*x*). Repetition is easy to show, the constants are repeated – for example (*a*, *b*, *a*). For order, we replace the comma delimiters with the less-than sign symbol '<' introduced earlier: for example (*a*<*b*<*c*), is read as (*a then b then c*).

However, even our small range of kinds has further formal requirements for its inputs. As Fine [6, p. 567] says, what he calls the 'logical form' of the application of the compositional operations needs to be permissive, allowing for variably polyadic as well as sensitive to order. It also needs to be sometimes sensitive to repetition.

One can enumerate these requirements. We have already discussed the application of zero objects – where the constructor generates a null object of the appropriate kind. This suggests an application requirement for an empty plurality – '()' in our notation. This is non-standard [33], but there are examples [36, pp. 154–5]).

The Absorption and Permutation principles lead to application requirements for repetition and order. One obvious way to do this is to allow for repetitive and ordered pluralities (though it is not agreed whether these appear in natural language [37, 38]) giving us a taxonomy of pluralities in Figure 10.

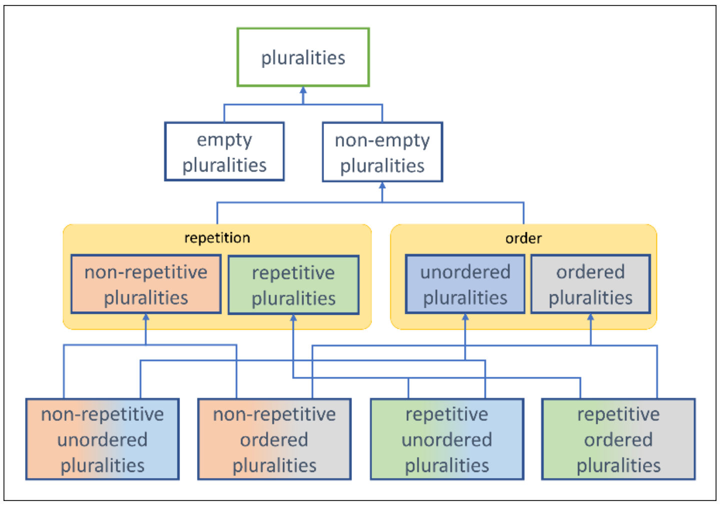

pluralities
empty pluralities
non-empty pluralities
repetition
non-repetitive pluralities
repetitive pluralities
order
unordered pluralities
ordered pluralities
non-repetitive unordered pluralities
non-repetitive ordered pluralities
repetitive unordered pluralities
repetitive ordered pluralities

Figure 10 – Taxonomy of pluralities

There is a more operational way to handle this. In the ontological framework, the metaphorical genie applies a constructor in all possible ways until the possibilities are exhausted. One way of doing this is to provide the genie with the available objects (as a plurality) and let it pick and provide all the relevant possible arrangements of these needed to exhaust the constructor – as a kind of batch process. In this way, all varieties in Figure 10 becomes ways in which the operation organises an initial input, rather than different kinds of plurality. So only simple pluralities are input.

Returning to the STRING-EXAMPLE-1: <(*a*, *b*, *c*), (STRING-BUILDER)> ontology, which you recall adopts the Absorption principle – so is just sensitive to order not repetition. Then the genie takes the input and, recognising the requirements of the constructor, works out all the possible ways these objects can be selected and ordered, which are:

$a<b$, $b<a$, $a<c$, $c<a$, $c<b$, $b<c$ and so on – see Figure 8

And simultaneously applies these to the STRING-BUILDER constructor.

We call this operation POWER, given its similarity to the initial stages of the powerset axiom when used in conjunction with SET-BUILDER. (Fine [24, p. 93]) introduces an operation called POWER for constructing sets from pluralities.) Then we can write the example as:

(*a*, *b*, *c*) POWER (STRING-BUILDER)

One outcome of this approach is that in ontogenesis, the POWER wrapped constructor is a unit of work – bare constructors without their POWER wrappers are incomplete, as they have no mechanism to add order and repetition to the simple pluralities.

Outputs are just the objects constructors themselves (so not a plurality), though there may be many of them – either from a single constructor (for example, THING-DIVIDER) or a POWER operation. If there is a sequence or chain of operations where the output of one is input to another. Then outputting operation constructs objects and the inputing operation collects these into a plurality for processing.

3.5.2 Stages

Constructional systems evolve in stages: Conway's Game of "Life" is built through simultaneous executions called generations – that both add and remove objects. Boolos [23] developed a stage theory – so-called as it generates sets in cumulative, expansive stages. These stages arise naturally out of a constructional framework. At each stage there is an existing population of objects (the expanding object domain at that stage) and possible constructions given that population. Executing all these constructions is a stage, which can potentially change the population, adding or removing objects. If this happens, a new population emerges, which, if there are possible constructions, is fodder for the next stage. Each stage is indexed by an ordinal, starting with stage 0 – which in our constructional ontology is the introduction of the givens.

Fine [5] produces a stage theory in this mould; developed in [6] for the composition constructors (In the latter paper, he adopts a sense of level – p. 584 "The *level* of an object is the first stage at which it appears" – that we don't use in this paper; as noted earlier we use the MLM sense of 'level').

We can define these stage operations for an individual constructor using the POWER operation described earlier. Assume we have an ontology with nominally givens *gg* (at stage 0) and a single constructor cr1, then each stage operation has the form:

(*xx*) POWER (cr1)

Where the *xx* are the plurality of objects available for application at that stage

We introduce the operation STAGE, which repeats – and at each repetition stage it applies the plurality of objects available for application at that stage. We can then write this ontology's ONTOGENESIS as:

STAGE [N]*

(*xx* [N]) (POWER (cr1)) => generation [N]

Where the *xx* [N] are the plurality of objects available for application at stage N

With generation 0 = givens

One might have thought that the *xx* [N] would just be the population, the whole object domain, at that stage. A later section looks at the conditions of application that explain why things are not that simple.

3.5.3 Expanding levels of objects

In the *scaling down* trope there is a narrative of starting with a *ubiquitous* pattern and finding an explanation through *scaling down* – as well as unearthing new patterns. We now work through an example of this. MLM has identified a multi-level type pattern. We explain below how the operation of constructors that reject the Levelling principle builds these kinds of multi-level hierarchies.

Fine [24, pp. 100–3] looks at the general principles for the way in which construction expands the constructed domain with new objects. When one starts looking at how composition constructors expand the domain, we find that for some constructors the expansion is levelled and for others it is flat. The key principle is Levelling. For the things and strings, where this principle is adopted, the execution of the constructors expands the domain, but remains flat without levels. For the sets and sequences, where the Levelling principle is rejected, the stages not only expand the domain but also a new level.

Where Levelling is adopted, the principle is; "components of composite have components, these components' components are also components of the composite." Let's define a component-component as a component of a component. The Levelling says all a composite's component-components are also components. In this case, component-components are at the same level as the composite – no new level is introduced.

Where Levelling is rejected, then some component-components are not components. (We need to introduce the qualification 'some' as one can devise exceptions.) In this case, both the components and their component-components are at different lower levels to the composite.

Here is an example ontology. Assume an ontology SOCRATES-EXAMPLE: <(*Socrates*), (SET-BUILDER)>. Ignoring the details of ontogenesis, let's consider the repeated application of SET-BUILDER to the given *Socrates*, the outcome is shown in Table 9. This clearly shows that there are cases where the component-component is not a component. At each stage a new generation of constructees are generated, and these constitute the next level.

Table 9 – Example Constructees Generated

Stage	**Given**	**Example Constructees Generated**	**Constructee components**	**Constructee component-components**	**Is the component-component a component?**
0	Socrates				
1		{Socrates}	Socrates		
2		{{Socrates}}	{Socrates}	Socrates	No
3		{{{Socrates}}}	{{Socrates}}	{Socrates}	No

This gives us our example of the *scaling down* narrative. We start with MLM's multi-level-types pattern. We provide a constructional explanation of it. These types have a similar CLAP profile to sets. As they reject Levelling, their ONTOGENESIS constructs a new level for each stage. These levels are a reflection of the stage they were constructed. We also have a new pattern – or rather a more general pattern. Multi-levelness extends beyond types to any Levelling-rejecting constructor, including in our compositional scope, sequences. We return to this later.

3.5.4 Populations and generations

We now look at another example of the *scaling down* trope. We start with a ‘level-respecting’ principle identified by the MLM community. We explain how it arises constructionally through a choice made on the condition of application.

Multi-level-types come in two forms, which are differentiated by whether types of one level can mix with types of another level. For one kind of type, types of one level do not mix with types of another level – which means that the instances of the types are always of the level below. For the other kind of level, this constraint is completely relaxed, and the instances of types can be from any previous level. Atkinson and Kühne [39] distinguish the kinds by whether they follow ‘strict metamodelling rules’. Kühne [40] by whether they follow a ‘level-respecting’ principle. Here we shall talk about pure and mixed levels.

This is not an isolated MLM pattern, it also appears in mathematics, where there is a similar significant differentiation between type and set theory. Russell’s Simple Type Theory (introduced in “Appendix B: The Doctrine of Types” of [41] and its descendants are strictly stratified – their types are pure and do not mix. Using Kühne’s term, they are level-respecting. Whereas set theory (first axiomatized in [42], has no such restrictions and so includes mixed sets. Gödel [43], as discussed in [44], claims that set theory “is nothing else but a natural generalization of the theory of types, or rather, it is what becomes of the theory of types if certain superfluous restrictions are removed.” Where one of the main restrictions is Kühne’s level-respecting principle. Boolos [23] provided a constructional (cumulative) framework which he called stage theory for mixed sets.

The difference between pure and mixed levels can be explained constructionally by different rules for what can be applied at a stage. Let’s take the pure ‘level respecting’ case first. Constructionally this happens when only the objects constructed in the previous generation are input into the next stage’s constructors – objects from earlier generations are excluded.

This example illustrates the pattern. Assume the formula for an ontology GENERATION-FORMULA1: <(*gg*), (CR-GEN)>, where CR-GEN is a levelling-rejecting constructor. Assume that the conditions of application for this stipulate that only the objects generated at the previous stage (its generation) are available to the next generation. The first few stages and a proforma *N*th stage are shown in Table 10.

Table 10 –GENERATION-FORMULA1: stages, inputs, generations and populations

Stage	**Input**	**Generated**	**Population**
0			population 0 (= GIVENS)
1	(GIVENS)	generation 1	population 1 (= generation 1 + population 0)
2	generation 1	generation 2	population 2 (= generation 2 + population 1)
3	generation 2	generation 3	population 3 (= generation 3 + population 2)
N	generation N-1	generation N	population N (= generation N + population N-1)

We can add this notion of a generation – all the (new) elements generated at a stage – to the stage formula and write GENERATION-FORMULA1's ONTOGENESIS as:

STAGE [N]*

(generation [N-1]) (POWER (CR-GEN)) => generation [N]

With generation 0 = givens (*gg*)

This may be clearer with a concrete example. Let's consider a SOCRATES-GENERATION-EXAMPLE: <(Socrates), (SET-BUILDER)> ontology. This an extension of the earlier SOCRATES-EXAMPLE, where we make explicit the commitment to applying generations in ONTOGENESIS. To create more space, we abbreviate *Socrates* to *S* in this and future similar examples.

STAGE [N]*

(generation [N-1]) (POWER (SET-BUILDER)) => generation [N]

With generation 0 = *S* (*Socrates)*

This evolves as shown in Table 11.

Table 11 – SOCRATES-GENERATION-EXAMPLE: stages, inputs, generations and populations

Stage	Input	**Generated**	**Population**
0			*S*
1	*S*	{*S*}	*S*, {*S*}
2	{S}	{{*S*}}	*S*, {*S*}, {{*S*}}
3	{{*S*}}	{{{*S*}}}	*S*, {*S*}, {{*S*}}, {{{*S*}}}

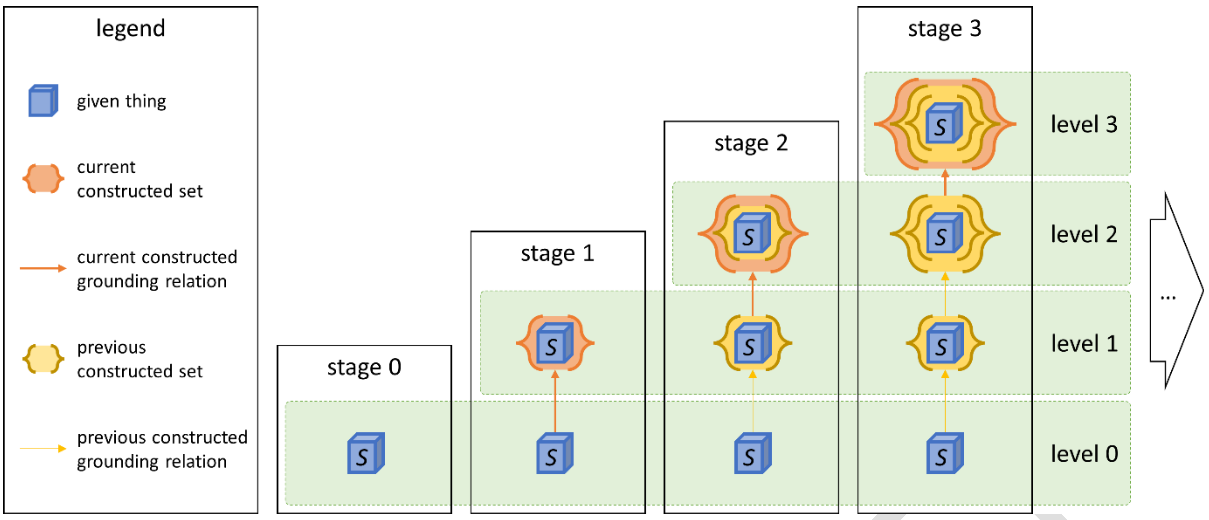


Figure 11 – SOCRATES-GENERATION-EXAMPLE – stage visualisation

When an ontology is constructed with these generation stages it will be pure.

To construct a mixed ontology, we change the rules of application. We apply all the existing objects, all objects from all previous generations (including GIVENS, generation 0) – what we call populations – at each stage.

Let’s illustrate this formula for an ontology POPULATION-FORMULA2 with givens (*gg*) and a single levelling-rejecting constructor CR-POP – where CR-POP takes population rather than generation stages. The first few stages and a proforma *N*th stage are shown in Table 12. As the input to later stages is larger (as it is the whole population and not just the previous generation) the contents of the later generations are also correspondingly larger.

Table 12 – Population-based POPULATION-FORMULA2

Stage	**Input**	**Generated**	**Population**
0			population 0 (= GIVENS)
1	population 0	generation 1	population 1 (= generation 1 + population 0)
2	population 1	generation 2	population 2 (= generation 2 + population 1)
3	population 2	generation 3	population 3 (= generation 3 + population 2)
N	population N-1	generation N	population N (= generation N + population N-1)

and its ONTOGENESIS is written as:

STAGE [N]*

(population [N-1]) (POWER (CR-POP)) => (generation [N])

With generation 0 = *Givens*

Where (population [N]) = (population [N-1]) + (generation [N])

(We shall omit the note that '(population [N]) = (population [N-1]) + (generation [N])' from future formulae as it is generally true.)

Consider an ontology SOCRATES-POPULATION-EXAMPLE: <(*Socrates*), (SET-BUILDER)> similar to the earlier Socrates ontology, but with the population conditions of application. Its ontogenesis is:

STAGE [N]*

(population [N-1]) (POWER (SET-BUILDER)) => (generation [N])

With generation 0 = *S* (*Socrates)*

This evolves in a different way, as shown in Table 13 and Figure 12.

Table 13 – SOCRATES-POPULATION-EXAMPLE: stages, inputs, generations and populations

Stage	**Input**	**Generation**	**Population**
0			*S*
1	*S*	{*S*}	*S*, {*S*}
2	*S*, {*S*}	{{*S*}}, {*S*, {*S*}}	*S*, {*S*}, {{*S*}}, {*S*, {*S*}}

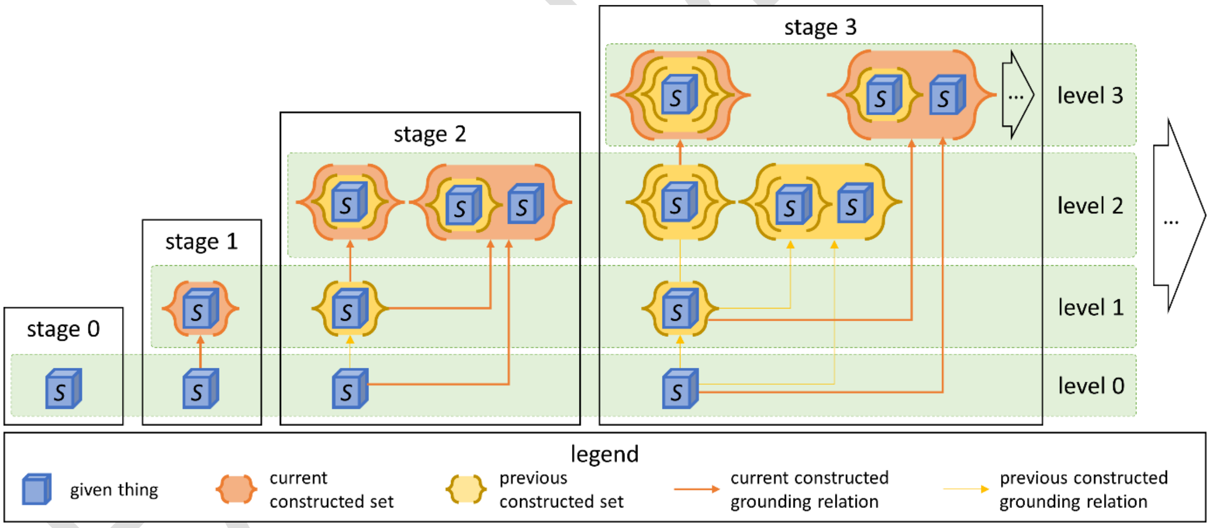
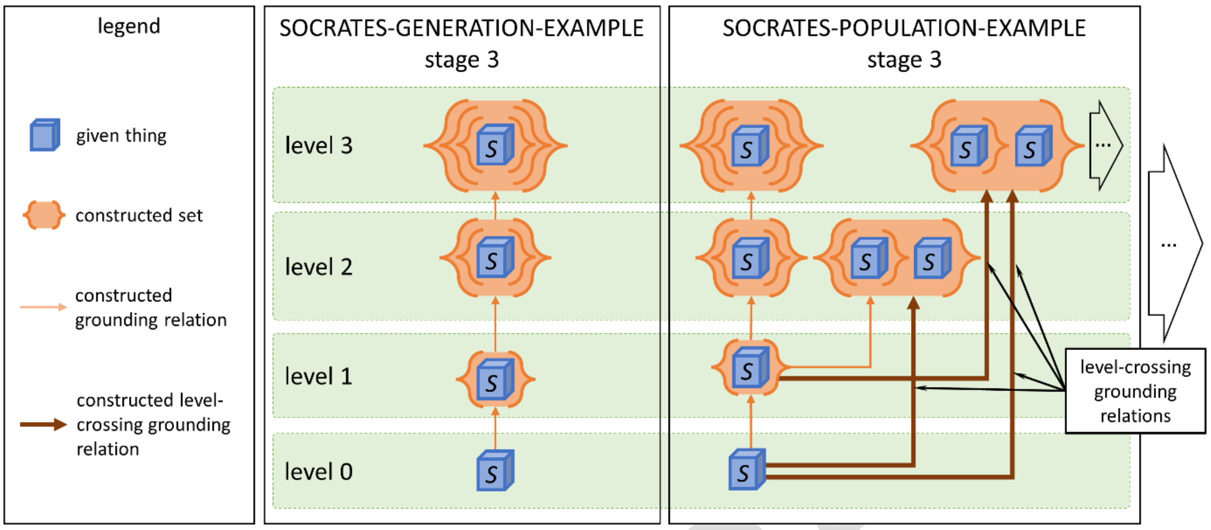


Figure 12 – SOCRATES-POPULATION-EXAMPLE – stage visualisation

Figure 13 compares stage 3 of the generation and population variants of the Socrates example. We can see how the links in generation application are only from one level to the next, whereas the links in population application cross levels.



legend
given thing
constructed set
constructed grounding relation
constructed level-crossing grounding relation
SOCRATES-GENERATION-EXAMPLE stage 3
SOCRATES-POPULATION-EXAMPLE stage 3
level 3
level 2
level 1
level 0
S
...
level-crossing grounding relations

Figure 13 – Socrates examples – generation and population stage 3 comparison

The operational view gives a clear picture of how the generation and population stage hierarchies emerge from a choice of which objects are fed into the genie's hopper for the generation of the next stage. One can also see that where two constructors only differ in terms of generation and population application, that the population's object domain is an expansion of the generation's object domain (as any object constructed by the generation will also be constructed by the population – see Figure 13). From our perspective, one would need to justify why one restricts this expansion. Without a good justification, given a choice, we (like Gödel [43]) see the level-respecting principle as a superfluous restriction and so elect for ontologies with population stages.

3.5.5 Principle of Plenitude

There are many ways structures can be restricted. Levelling-rejecting sets and sequences have few or any constraints upon which types of object can be components. Other structures are by comparison, much more restrictive. For example, first-order logic won't allow predicates to be predicated resulting in an unmixed two-level structure – higher-order logic lifts the restriction. UML has a similarly restrictive structure.

In our view, foundational ontologies need to be expressive and we aim to keep restrictions to a minimum. This is a kind of Lewisian principle of plenitude [45, pp. 86–92] with a preference for abundance over sparsity. Usually, if it is possible to be more expressive, then there needs to be a good reason why not. If one has a requirement for sparsity, one can deal with it down the line.

Another constructional choice between abundance and sparsity we face arises over the question of whether to segregate the levelling-rejecting-species (sets and sequences). Should objects of all other species be allowed to be applied to the constructors of other species. One could imagine an ontology where sequences could not be members of sets and sets could not be members of sequences. This is easy to arrange constructionally, one just restricts what kinds of object can be applied to the relevant constructors. The orthodox sets and sequences mix species, allowing sequences to be members of sets and sets members of sequences. However, unorthodox sparse constructors are possible and easy to construct – as is illustrated by the two edge choice types in Table 14. Given our attitude to plenitude, we choose maximal abundance – the orthodox choice.

Table 14 – Edge abundant and sparse choice types

Constructor	Choice type	Input choices
SET-BUILDER	minimally sparse	set, thing
SET-BUILDER	maximally abundant	set, thing, sequence, string
SEQUENCE-BUILDER	minimally sparse	sequence, thing
SEQUENCE-BUILDER	maximally abundant	sequence, thing, set, string

From a population stage management perspective, this means as sets and sequences are generated at one stage, they are ready to be fed into both sets and sequences at the next stage. The ONTOGENESIS is quite simple:

```
STAGE [N]*
        (population [N-1])
                POWER (SET-BUILDER)
                POWER (SEQUENCE-BUILDER)
        => (generation [N])
```

Within the stage, the constructors can execute independently, across stages, their outputs merge.

3.5.6 Building an ontology: choosing constructors

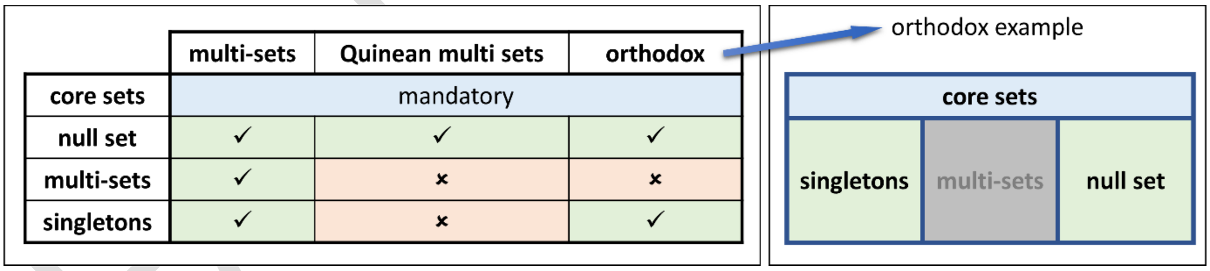
We have restricted ourselves, as discussed above, to the four kinds of object. When setting up an ontology, one has a choice of which of these four to include in the constructor domain. The choice of kinds seem to be independent: for example, choosing to include SET-BUILDER does not seem to either exclude THING-BUILDER or SEQUENCE-BUILDER or force their inclusion. Making this choice sets the kind-scope of the ontology.

Each kind comes with a family of potential constructors – giving a total of fourteen constructors. Having set the kind-scope, for each kind in scope one needs to decide what constructors to include in the domain. Prima facie it seems these are also independent, that one has a choice for each of the fourteen whether to include this in the constructor domain. For example, if SET is in scope, then one can choose whether to include both the orthodox and the multi-set constructors. However, this may not be the best way to formulate the choice.

Firstly, while we have established that kinds have disjoint identity, we need to look at identity with the kind families of constructors. Consider this case. Allow two SET-BUILDER constructors into the domain, one that allows empty inputs and so generates the null set, and the other that does not; SET-BUILDER-EMPTY and SET-BUILDER-NON-EMPTY. This leads to an odd but not inconsistent situation. Apart from the null set, both constructors will generate the same sets – assuming extensional identity for the sets. Hence, the ontology would be the same if just SET-BUILDER-EMPTY was included – SET-BUILDER-NON-EMPTY is superfluous.

A similar situation holds for Absorption and multi-sets. If one has two SET-BUILDERs, one allowing repetition the other not, then the constructor not allowing repetition is superfluous. Everything it constructs, the other constructor constructs as well. Again, one choice subsumes the other. So here the choice is really whether one wants multi-sets as well as orthodox sets. And also, for Collapse and Quinean sets, the orthodox constructor will generate singletons, the Quinean constructor will not. One needs to decide whether to include singletons.

In each case, one choice subsumes the other. The choices boil down to what to include in the ontology, to expand or contract the range of the ontology. It is much easier to see this as, given the kind constructor SET-BUILDER, making choices of disjoint optional sub-species. The variety arises from the choice of which optional sub-species are included. One can visualise this as ticking rows in a table or choosing the optional sub-species to include – as shown in Figure 14.



	multi-sets	Quinean multi sets	orthodox
core sets	mandatory		
null set	✓	✓	✓
multi-sets	✓	×	×
singletons	✓	×	✓

Figure 14 – Visualising selecting a variety – and an example of choosing the optional sub-species

Hence, it is maybe better to adopt as a general policy: that the different CLAP profiles and conditions of application for each kind are ways the single constructor can be configured rather than different constructors in their own right. So there are four base constructors with a variety of configurations. This will help one avoid the temptation to include multiple constructors for a kind in the domain.

3.5.7 Constructor Staging Options

Strict staging occurs where there are no constraints upon the metaphorical genie executing a possible operation. At the very first stage a construction is possible, it is executed. Where there is more than one constructor, it can be easier to visualise the execution if the stages are split by constructor.

Consider this SPLIT-STAGING-EXAMPLE: <(c), (THING-DIVIDER, SET-BUILDER> ontology, where the given c is the mereological fusion of two thing atoms a and b. Table 15 shows the strict stages. One can see that the singletons $\{a\}$, $\{b\}$ and $\{c\}$ are split over two stages, in a way that does not happen where there is a single SET-BUILDER constructor. The indexing of the stages no longer matches the indexing of the levels (all singletons are same level in the MLM sense of level).

Table 15 –SPLIT-STAGING-EXAMPLE – strict staging

Stage	**Input**	**Constructor**	**Generated**	**Population**
0				c
1	c	THING-DIVIDER	a, b	$a, b, c, \{c\}$
		SET-BUILDER	$\{c\}$	
2	$a, b, c, \{c\}$	THING-DIVIDER		$a, b, c, \{c\}, \{a\}, \{b\}, \{a, b, c\}, \{\{c\}\}, \{a, b, \{c\}\}$, etc.
		SET-BUILDER	$\{a\}, \{b\}, \{a, b, c\}, \{\{c\}\}, \{a, b, \{c\}\}$, etc.	

Table 16 shows what happens if the staging is split by constructor. Where THING-DIVIDER is exhausted first, and once exhausted, the SET-BUILDER stages start. Here the singletons are in a single stage – and more generally, the indexing of the stages matches the indexing of the levels.

Table 16 – SPLIT-STAGING-EXAMPLE – split staging

Stage	**Input**	**Constructor**	**Generated**	**Population**
0				c
1	c	THING-DIVIDER	a, b	a, b, c
2	a, b, c	SET-BUILDER	$\{a\}, \{b\}, \{c\}$	$a, b, c, \{a\}, \{b\}, \{c\}$
3	$a, b, c, \{a\}$, {b}, {c}	SET-BUILDER	$\{\{a\}\}, \{\{b\}\}, \{\{c\}\}, \{a, b, c\}, \{a, b, \{c\}\}$, etc.	$a, b, c, \{a\}, \{b\}, \{c\}, \{\{a\}\}, \{\{b\}\}, \{\{c\}\}, \{a, b, c\}, \{a, b, \{c\}\}$, etc.

One can visualise stage splitting as partitioning the constructors and then partially ordering them. Then the stages are executed for each partition and the output is, in effect, the givens to the next partition stages. In this case, the two constructors are each given their own partition, and the outputs from the THING-DIVIDER partition are the givens for the SET-BUILDER partition. With more constructors, richer partial orderings are possible.

As the example shows it makes no difference to the final object domain whether split or strict staging is adopted. (Though one need to be careful how one splits the staging, so that this is the case.) But there is one important benefit. Split staging, by stretching out strict staging, allows levels (in the MLM sense) to match with stages, providing a neater, more parsimonious organisation (there is no need to explain a disjunction between stages and levels). When we build the constructional BORO Ontology below, we will adopt a split staging for THING-DIVIDER for this reason.

In a way, splitting is a recapitulation inside a single ontology of multiple ontologies being merged in ontological space. Indeed, the split we have been looking at would naturally occur as multiple ontologies merging into one in a sandpit ontological space. We have focused on the way splitting allows to us keep a clean match between stages and levels. But more generally, splitting (like sandpit ontological spaces) makes the underlying structures and their relationships more perspicuous.

3.5.8 Splitting SET-BUILDER – kind levels

The orthodox varieties of the levelling-rejecting constructors are unrestricted in what kinds can be applied. When they generate their levelled hierarchies, these have components from all the allowed kinds. However, by splitting the stages, one can build levelled hierarchies based upon the kinds.

Here is an example where we split sets based on things from sets based upon sequences of things. Take the ontology SPLITTING-SET-BUILDER: <(*thing*), (THING-DIVIDER, SET-BUILDER, SEQUENCE-BUILDER)>, where *thing* is some given thing. At this level of generality, it does not matter whether these have a generation or population applications. For each constructor we fabricate a standard stage process formula which takes a variable given domain and exhausts the constructor and outputs the (complete) population; EXHAUST-THING-DIVIDER, EXHAUST-SET-BUILDER and EXHAUST-SEQUENCE-BUILDER.

We start by applying the given *thing* to EXHAUST-THING-DIVIDER outputting the population *things*. This gives us the basis for the split. We then apply *things* independently to both EXHAUST-SET-BUILDER and EXHAUST-SEQUENCE-BUILDER outputting respectively the populations – *things-sets* and *things-sequences*.

We then repeatedly apply EXHAUST-SET-BUILDER separately to each exhaust output population and EXHAUST-SEQUENCE-BUILDER to pure set exhaust output populations. Finally, we apply the full strict process to generate the objects that have not been constructed by the split process. Figure 15 shows the split pre-strict process. This shows *things* and three expanding series. It divides these into two hierarchies each with two groups (groups are shown in different colours in the figure); one based upon sets of things with no sequences, another composed of sequences and sets of sequences.

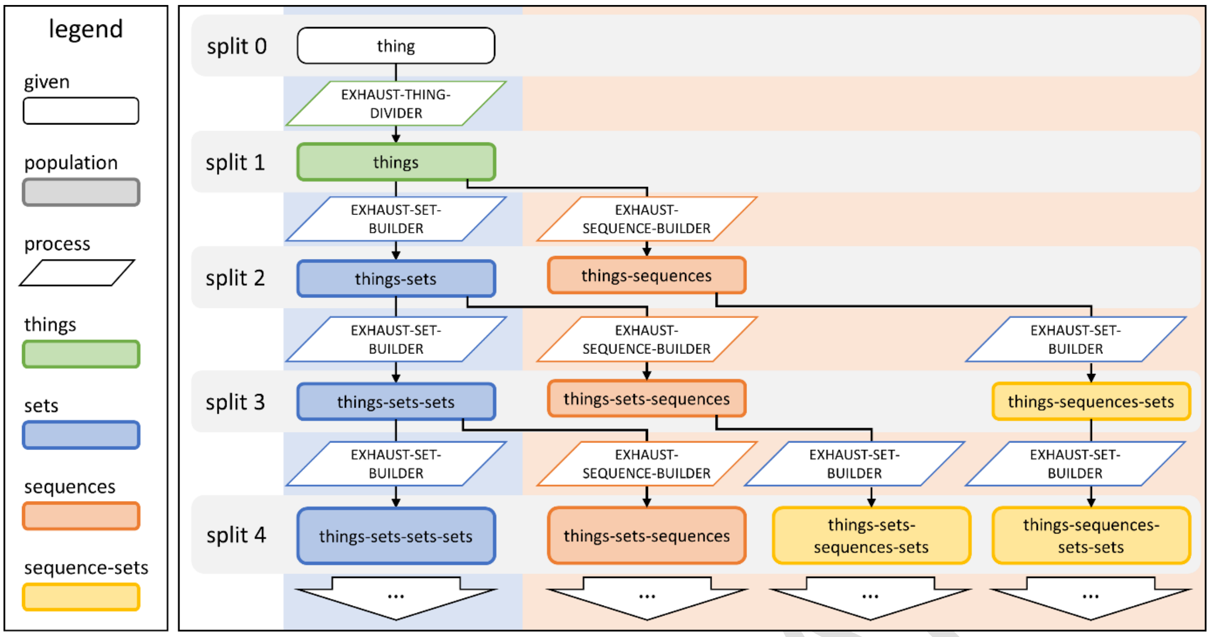


Figure 15 – Splitting (exhaustion) process

These formal structures correspond to entities and relation in entity relationship modelling. Things are entities, sets of things are entity types, thing sequences are individual relations and sets of thing sequences are relationships. This neatly illustrates how 'playing' with the stage process can reveal interesting structures.

3.6 *Basic and derived constructors*

[6, pp. 576–8] suggests that a composition relation is basic if it cannot be constructed from other constructors. If it can be constructed, then it is derived. One can also regard the constructional derivation of constructors as another example of how *scaling down* explains *ubiquitous* patterns – in particular showing they are derived rather than foundational.

Fine offers as an example set-theoretic union as the basis for the sub-set relation that has a THING-like CLAP profile. He notes this is an example of the more general case of flattening a multi-level hierarchy – reversing the effects of rejecting the Levelling principle. He offers a mathematical recipe for the flattening of unions, here we provide a constructional explanation of the underlying relation as this fits better with our operational theme.

3.6.1 *Deriving SUBSET-BUILDER – flattening SET-BUILDER*

To derive SUBSET-BUILDER, we need to introduce the converse of SET-BUILDER, CONVERSE-SET-BUILDER. This unwinds the set construction, taking the set and constructing its members. It is dependent upon the original set construction. One can imagine the metaphorical genie remembering how the set was constructed and so knowing what the members are.

Consider SET-EXAMPLE-1: <(a, b, c), (SET-BUILDER)> ontology and we are at a stage in ONTOGENESIS where the genie has constructed the set {a, b, c}. If we apply {a, b, c} CONVERSE-

SET-BUILDER we construct the plurality of its members (*a*, *b*, *c*). If we now apply (*a*, *b*, *c*) POWER (SET-BUILDER) we construct all the sub-sets of the original set; {*a*}, {*b*}, {*c*}, {*a*, *b*}, {*a*, *c*}, {*b*, *c*} and {*a*, *b*, *c*}.

We generalise this to the aggregate constructor relation, (*x*) SUBSET-BUILDER, which is fabricated from basic constructors as follows: ((*x*) CONVERSE-SET-BUILDER) POWER (SET-BUILDER). This constructs the sub-sets for a set.

SUBSET-BUILDER has CONVERSE-SET-BUILDER as a component, and this component depends upon SET-BUILDER, so SUBSET-BUILDER does too. Hence if SUBSET-BUILDER is in the constructor domain, then SET-BUILDER needs to be too. But, obviously, CONVERSE-SET-BUILDER itself does not need to be – it is just smuggled in for a restricted set of operations inside SUBSET-BUILDER.

3.6.2 SUBSET-BUILDER: ONTOGENESIS levelling

It is worth taking a little time to see how SUBSET-BUILDER fits into ONTOGENESIS. We treat the derived constructor in the same manner as the basic constructors, which gives us:

```
STAGE [N]*
        (population [N-1])
                POWER (SET-BUILDER),
                POWER (SUBSET-BUILDER)
                => (generation [N])
```

At stage N, all the sets constructed by stage N-1 will be applied to POWER (SUBSET-BUILDER) and their sub-set relations will be constructed. Sets constructed at stage N will have their sub-set relations constructed at stage N+1.

One can begin to see the structural similarities between SUBSET-BUILDER and THING-DECOMPOSER noted in [46]. And it is no surprise they have the same CLAP profile (**CLAP**). For example, they both adopt Levelling and so have a flat structure. However, the constructional approach makes clear, as [6, p. 580] notes, that THING-DECOMPOSER is basic and the SUBSET-BUILDER is derived.

3.6.3 Flattening SEQUENCE-BUILDER

A similar story can be told for the flattening of SEQUENCE-BUILDER by fabricating the derived SUB-SEQUENCE-DECOMPOSER as CONVERSE-SEQUENCE-BUILDER.POWER (SEQUENCE-BUILDER). The need to handle order and repetition introduces a wrinkle. When the genie executes CONVERSE-SEQUENCE-BUILDER, it needs to take note of the order and repetitions, so that when POWER submits the various objects to SEQUENCE-BUILDER, this is done in a way that respects that order and repetition. In cases like this, the exotic taxonomy of pluralities discussed earlier might be useful.

3.6.4 Deriving POWERSET

One can regard the constructional derivation of powerset as another example of how *scaling down* explains *ubiquitous* patterns. Zermelo-Frankel set theory postulates the existence of the powerset in the powerset axiom [42] at the core of set theory – with the implication it is fundamental. Powersets are (like multi-level-types, to which they are related) ubiquitous in information systems, see [10].

Boolos [23, p. 225] describes how he derives the powerset axiom in his cumulative (constructional) set theory. He considers the stage *t* at which a set *z* is formed and *s* the next stage. All the sub-sets of *z* are formed at stages before *s*. At *s* there is a new plurality of the sub-sets of *z* (which includes *z*) from which a new set, the powerset is formed. This clearly shows that the powerset is generated one stage later than its underlying set. Also, every non-empty set formed at a stage *n* will have its powerset formed at the next stage *n+1* – though not every set formed at stage *n+1* will be a powerset. From the cumulative perspective, powerset may be useful but it is derived, not fundamental.

Here we describe how to derive powersets constructionally. Given we have derived SUBSET-BUILDER, we use it to fabricate a POWERSET-BUILDER constructor. Its internal form is:

((*x*) SUBSET-BUILDER) SET-BUILDER => powerset [*x*]

Or dismantling SUBSET-BUILDER

(((*x*) CONVERSE-SET-BUILDER) POWER (SET-BUILDER)) SET-BUILDER

=> powerset [*x*]

The grounding relation between the input set and output set is powerset.

This gives us two equivalent ways of deriving POWERSET-BUILDER. We take a set as given. We can either take the pluralities output from applying SUBSET-BUILDER (all the sub-sets of the initial set) and then build a set from them. Or we can use CONVERSE-SET-BUILDER to construct the plurality of all the members of the given set, then apply this to POWER (SET-BUILDER) – generating a plurality of sets – and then building the set that has this last plurality as members.

Some care needs to be exercised when adding this constructor to ONTOGENESIS to ensure that the conditions of application are staged properly. When POWERSET-BUILDER is executed at stage N, only sets from stage N-1 should be allowed as input. Assuming a simple ontology with SET-BUILDER and POWERSET-BUILDER the ONTOGENESIS is of this form:

STAGE [N]*

(population [N-1]) POWER (SET-BUILDER)

(population [N-2]) POWER (POWERSET-BUILDER)

=> (generation [N])

Where population [0] = Givens

As there is no dependence, the genie can run the two intra-stage POWER operations simultaneously – on different pluralities.

3.6.5 Deriving stage-sets and stage-powersets

The constructional approach also reveals the role powerset plays in linking stage populations and generations. In this approach, the population of the ontology expands at each generative stage as illustrated in Figure 16, which translates into the stage-plurality-formula: (population [N]) = (population [N-1]) + (generation [N]). The figure is neutral on whether population or generation application conditions have been chosen, as these both have the same topological structure – but as already noted, metrically, a choice of population typically leads to an accelerated expansion with a greater range of mixed objects.

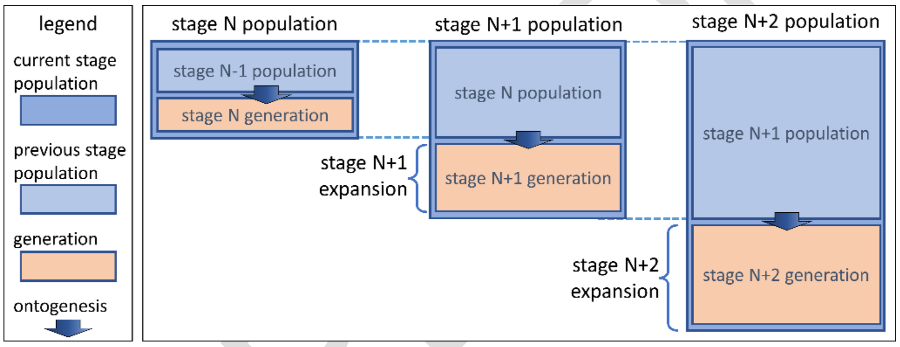


Figure 16 – Visualising expansion

However, the pluralities are creatures of the constructional process, they do not exist in the ontology. For his cumulative hierarchy, Boolos [23, p. 228] introduces the “sequence of sets, {R_α}, with which the stages of the stage theory may be identified”, where R_α is the set of the population at stage α. He then goes on to describe how the powerset can be used to recursively define the stages – where the sets of stage α+1 are the union of stage α’s population set and its powerset. Using objects in the ontology, sets, is useful as it enables one to talk about stages within the ontology – and also reveals the powerset relationship.

We can derive these sets constructionally. We do this, in two steps. Firstly, constructing the sequence of sets and then the power set relation.

Each stage constructs a plurality of objects – its generation. For ontologies that include SET-BUILDER, this plurality is constructed into a set – the stage set – at the next stage. One can formally recognise this by adding a derived constructor to the stage process:

STAGE [N]*

STAGE-SET: (population [N-1]) SET-BUILDER => stage-set [N-1]

Then at stage N, there is an ordered plurality of sets

(stage-set [0] ⊂ stage-set [1] ⊂ stage-set [2] ⊂ stage-set [3] … ⊂ stage-set [N-1]

We can recognise the powerset relation between the population stage-sets by adding this derived constructor to the stage process:

STAGE [N]*

STAGE-POWERSET: (stage-set [N-2]) POWERSET-BUILDER

=> stage-powerset [N-2]

Where stage-powerset [N-2] = generation [N-1].

Then the stage-set-formula counterpart of the stage-plurality-formula is:

stage-set [N] = stage-set [N-1] ∪ stage-powerset [N-1]

(Strictly, we should define the derived union (∪) operation, but this is trivial, so we leave it out.)

This shows how axioms such as the powerset axiom and the powerset relation between stages emerges from the simple operations of the basic constructors.

Figure 17 shows how these objects can be used to visualise the stage structure in a different way, with added optional callouts showing the population and generation classes.

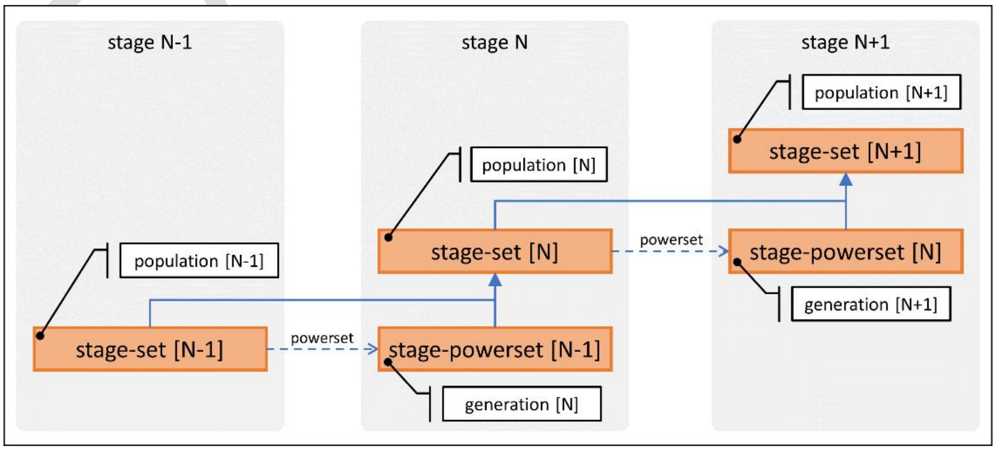


Figure 17 – Ontology's stage structure – powerset view

3.6.6 Deriving taxonomies

Subsets and powersets are general, foundational derived constructors. However, domain level hierarchies are common. Taxonomies (such as the Linnaean classification) and component breakdowns, in so far as they are ontological, are good examples. Deriving constructors for these domain hierarchies is usually simple; involving selecting a subset of an existing hierarchy – and this can be levelled in ways that the original hierarchy is not. One chooses a subset of the objects and then a subset of one kind of composition so that the result conforms to the desired structure; if one wants a levelled structure, then one ensures the structure conforms to the right CLAP principles – typically that it does not adopt Levelling. We look at how we can derive constructors for taxonomies in this section and for component breakdowns in the next section.

Consider (one version of) the Linnaean classification hierarchy that has 'Natural Things' at the top and is divided and sub-divided through the levels until reaching 'Felis leo' and 'Felis tigris' as the bottom. Make the sensible assumption that 'Natural Things' is a set of material objects and the other classifications are subsets of it [10]. This collection of classifications are the objects used in the hierarchy. These can be derived using SUBSET-BUILDER and a filter, FILTER-LC, that given a collection selects those elements that are Linnaean classifications; **LINNAEAN-C:** FILTER-LC (SUBSET-BUILDER (Natural Things)).

Traditionally, the classification structure is levelled by taking a transitive reduction of the underlying composition hierarchy. We can derive the LINNAEAN-SUBSET-BUILDER constructor for this by taking a constrained version of SUBSET-BUILDER that can only be applied to Linnaean classifications and when applied only reruns the next level – other levels are filtered out. This gives us a natural generation level hierarchy LC, which is defined as having 'Natural Things' as its given and LINNAEAN-SUBSET-BUILDER as its sole constructor – with the ontogenesis shown in Table 17.

Table 17 – LINNAEAN-SUBSET-BUILDER ontogenesis

Stage	Input	Stage Operations	Generated
0			(Natural Things)
1	(Natural Things)	LINNAEAN-SUBSET-BUILDER	generation 1
2	generation 1	FOR EACH X in Input LINNAEAN-SUBSET-BUILDER (X)	generation 2
…			
N	generation N-1	FOR EACH X in Input LINNAEAN-SUBSET-BUILDER (X)	generation N
…			

Of course, the classification structure is richer; which means more constructors are needed. For example, the levels (e.g. Kingdoms) are explicit as ranks (this is described in [10]), so a RANKER constructor is required. However, this skeleton outline should be sufficient to indicate how this could be done.

3.6.7 Deriving taxonomies breakdowns

The same derivation process can be used on other kinds of composition. Component breakdown structures (such as car X breaks down into body and engine components and engine into so on) would be based upon whole-part relations and THING-DECOMPOSER [47].

This ability to derive levelled hierarchies from the fundamental structures helps to explain (ontologically) what these hierarchies are and how the fundamental structures can be used.

3.6.8 Deriving MLM generalisation and classification hierarchies

The MLM community has analysed classification and generalisation and noted the similarities with, respectively, set-membership and subset: for example [48]. A common point made is that generalisation (like subset) is transitive and does not have levels whereas classification (like set-membership) is anti-transitive and has levels [40]. We have looked at the constructional versions of set-membership and subset – SET-BUILDER and SUBSET-BUILDER – and explained the relationship between the two. SET-BUILDER is a fundamental constructor, whereas SUBSET-BUILDER is a derived constructor – derived from SET-BUILDER.

SET-BUILDER and set theory considers every possible permutation of the members of a set as a subset. In UML and MLM, classification is not plain set-membership nor generalisation plain subset – and the differences can guide us on how to build the appropriate derived constructors. One key difference is that the conceptual models typically restrict their relations to a sub-domain of interest. As with the taxonomies and component breakdowns above, this can be captured by a filter on the constructor – where the filter clearly delineates the scope of the domain.

In UML and MLM there are many varieties of generalisation. In some MLM contexts, the generalisation hierarchy is, like in taxonomies above, restricted to a transitive reduction. This is also common in conceptual modelling contexts, where typically only the transitive reduction is modelled, and the transitive closure is rarely if ever mandatory. In some contexts, the hierarchy is restricted to a tree-structure, whereas in others it is more usual to allow lattice (multiple inheritance) hierarchies. Submitting these different structures to a sandbox analysis would capture their different commitments in derived constructors (typically using filters) with their appropriate CLAP profile – as well as their common underpinnings.

Similarly, there are varieties of classification. One we consider earlier is the ‘level-respecting’ property [40]. This turns on the conditions of application; whether the whole population is applied to a stage or only the previous generation. As this analysis showed, looking at these sorts of structures in terms of how they are derived from more fundamental constructors helps to both highlight and formalise their differences as well explain them.

It also gives rise to natural enquires as the motivations for the choices; for example, are they adopted for metaphysical/ontological or pragmatic reasons, and if so, what are these reasons? So, for example, it makes clear that adopting a 'level-respecting' or 'strict metamodelling' approach filters out mixed sets. One can ask what the motivation for this is and whether the cost of excluding them is worthwhile.

3.7 Grounding the framework

Grounding is an active area of research in philosophy. Here we are interested in a simple aspect of this. Schaffer's [21, p. 355] describes the structure of the Neo-Aristotelian ontology as "an ordered hierarchy generated from (i) a list of the substances F, plus (ii) a list of the grounding relations G" where the grounding relations reflect a dependence between the substances and so "generate the hierarchy of being". In our case, we have restricted our interest in dependence to composition. This allows us to frame the multi-level modelling community's engagement with multi-level-types and the relations of instantiation and generalisation as a project to understand an ordered ontology with compositional grounding relations.

In the framework, as it currently stands, the grounding relations are ways the process is executed, just relations between the inputs and outputs of constructors. If (*a*, *b*, *c*) SET-BUILDER is executed to construct {*a*, *b*, *c*}, then this implies *a*, *b* and *c* are members of {*a*, *b*, *c*} but, so far, there is nothing in the process to reify these ways in the ontology. They may be stored in the 'memory' of the metaphorical genie. But presumably, like a computer program, this memory is transient and disappears when the task is complete.

3.7.1 GROUND

We remedy this by introducing a new constructor, GROUND, that crystallises these metaphysical grounding relations in the ontology. From a process perspective, this is a wrapper that is put around all constructors that captures the structure of the process. It takes the input, applies it to the constructor. It takes the output and before passing it on creates the binary grounding relations. Adding this to the example above we get:

(*a*, *b*, *c*) GROUND (SET-BUILDER)

This not only generates {*a*, *b*, *c*}, but also three binary set-membership relations.

3.7.2 Generating grounding

It is worth noting that adding the grounding wrapper makes some previously merely re-constructing processes generative, in the sense that grounding objects are now generated. Returning to the earlier STRING-EXAMPLE-1: <(*a*, *b*, *c*), (STRING-BUILDER))> ontology, where the string *abc* can be built (constructed) in multiple ways; let's say by applying (*a*<*b*<*c*) or (*ab*<*c*) or (*a*<*bc*) in this order. Without grounding, only the first construction is generative. With grounding, then all the executions generate something, though some only generate grounding relations and not all the grounding relations are generated, some are only re-constructed.

3.7.3 Grounding relation roles

All grounding composition relations capture a relation between a composite and a component. As mentioned earlier, there is also the direction of operation, which can also be regarded as the direction of grounding. These don't always match. SET-BUILDER's direction of operation is component-to-composite, whereas THING-DECOMPOSER's is composite-to-component.

In this paper, there is not space for a detailed discussion, so we summarise the position. Relations can be structured in various ways [49]. In databases, the rows are often talked of as tuples – so based upon order. This is a standard mathematical approach. However, under the covers, they usually are role based (what Fine calls positional). Chen [50, p. 12] makes exactly this point, saying the ordering can be dropped once the roles are specified. One of the major differences between order and role-based relations is that order relations have a converse, whereas positional relations do not.

We have found that it seems to make more sense to treat grounding relations as role-based, where the role reflects both the composite-component and direction of operation facets. This gives rise to four concrete types of role as shown in Figure 18.

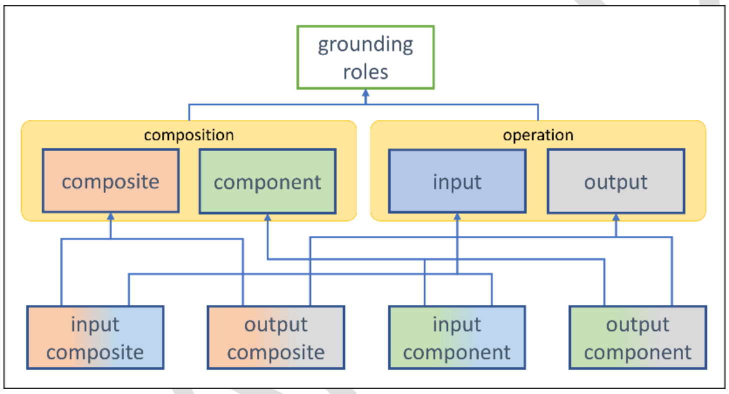


Figure 18 – Grounding roles taxonomy

3.7.4 Basic grounding relation types

The type of a basic grounding relation can be inferred from the kind of the objects occupying its positions. The types are as shown in Figure 19.

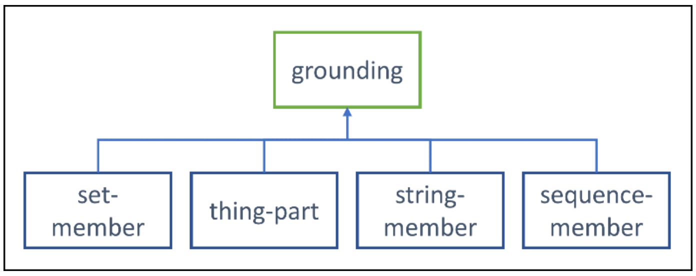


Figure 19 – Basic types of grounding relation

3.7.5 Grounding order types

For strings and sequences, the grounding relation needs to be sensitive to order and repetition. The simplest way to do this is to number the places in an order and let the grounding relation inherit these place numbers. Consider:

(*a*, *b*, *c*) POWER (GROUND (STRING-BUILDER))

Then one of the powered STRING-BUILDER executions generates the string *abc*. Its grounding wrapper generates three binary string placing relations – where the placing relation is typed with the number of the place it is linking So,

- String-Member Place1 (input-component (*a*), output-composite (*abc*))
- String-Member Place2 (input-component (*b*), output-composite (*abc*))
- String-Member Place3 (input-component (*c*), output-composite (*abc*))

The general typing scheme is shown in Figure 20.

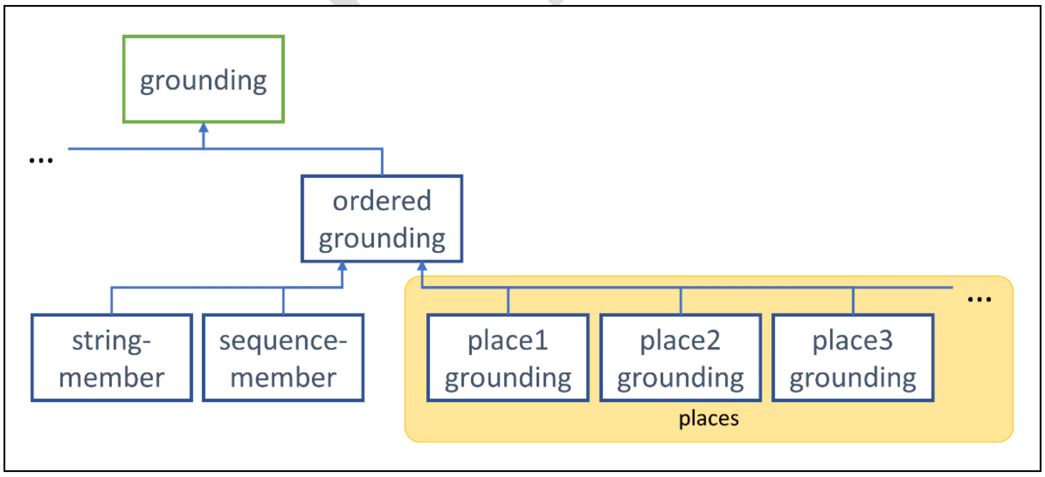


Figure 20 – General typing scheme

3.7.6 Grounding derived constructors

One needs to ground derived constructors such as SUBSET-BUILDER and POWERSET-BUILDER, to make the structure they construct visible. This is done in the same way as for basic constructers, by wrapping the constructor in GROUND. To make these derived relations identifiable, they need to be typed.

3.7.7 *Powerset-set – grounding identity*

The type structure for the basic grounding relations and their roles make it difficult to find examples where broad extensional identity is not fine-grained enough to ensure no questions of intra-species identity arise. However, the introduction of the derived Powerset construction raises an interesting question of intra-species identity. In standard set theory, and so mirrored in our constructional version, the powerset contains the original powered set as a member. This is because sub-set is reflexive, and so a set is a sub-set of itself and so included as a member of the powerset. This results in a powerset-set grounding relation that between the same objects as the set-member relation – it is identical, apart from its powerset typing. In the spirit of parsimony, one could allow a kind of intra-species identity, where the existing set-member relation is typed as a powerset-set relation. This would be justified, if we take the basic typing as the basis for a broad extensional identity. The result of this extension to the typing taxonomy is shown in Figure 21.

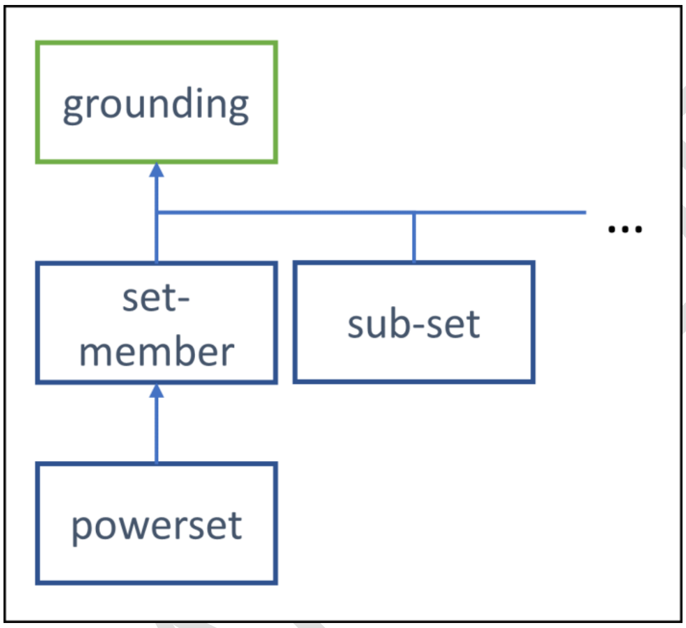


Figure 21 – The extension to the typing taxonomy

3.7.8 *First class grounding relations*

A second-class object [51] is one that is not given the same 'rights' as other objects. For example, from a population management perspective, where stages are generationally based, previous generations are treated as second-class citizens; they are not allowed to participate in construction.

There is a similar kind of choice of whether to segregate (stratify) grounding relations – making them second-class citizens. One could argue that one should not mix grounding relations and main kinds of objects, that they should be kept separate. Constructionally, one would then place a restriction on constructors that objects and grounding relations cannot be applied at the same time – or have two versions of the constructor. The two versions stage proforma for ONTOGENESIS for SET-BUILDER would look like this:

(*xx* objects) POWER (OBJECT-SET-BUILDER)

(*yy* grounding relations) POWER (GROUNDING-RELATION-SET-BUILDER)

Whereas, if mixing is allowed, the stage proforma is much simpler, though it is generatively richer – constructing far more (mixed) objects:

(*zz*) POWER (SET-BUILDER)

As noted earlier, our preference is for abundance over sparsity. If it is possible to have mixed sets – whether across stages or grounding relations – then why not.

To get a feel for the abundance, let us return to the *Socrates* example in Table 11 with added grounding. Here we adopt a simple notation for grounding using angle brackets and commas – where the first place is given to the component and the second to the composite – e.g. <component, composite>. The result is shown in Table 18 – where you can see it has many mixed sets.

Table 18 – POWER (GROUNDED (POPULATION-SET-BUILDER))

Stage	**Input**	**Generated**
0		
1	*S*	{*S*}, <*S*, {*S*}>
2	*S*, {*S*}, <*S*, {*S*}>	{{*S*}}, {*S*, {*S*}}, {*S*, <*S*, {*S*}>}, {*S*}, <*S*, {*S*}>}, {*S*, {*S*}, <*S*, {*S*}>} <*S*, {*S*, {*S*}}>, <*S*, {*S*, <*S*, {*S*}>}>, <*S*, {*S*, {*S*}, <*S*, {*S*}>}, <{*S*}, {{*S*}}>, <{*S*}, {*S*}, <*S*, {*S*}>}>, <{*S*}, {*S*, {*S*}, <*S*, {*S*}>}>, <<*S*, {*S*}>, {*S*, <*S*, {*S*}>}>, <<*S*, {*S*}>, {*S*}, <*S*, {*S*}>}>, <<*S*, {*S*}>, {*S*, {*S*}, <*S*, {*S*}>}>

4 A simple constructional BORO ontology

We now describe a simple example ontology, called (unimaginatively) SIMPLE based upon the BORO Foundational Ontology. It has been designed to get the right balance between being sufficiently simple to be easy to understand and being able to generate most of the complex patterns used both by the coordinates system case (which is looked at later in the paper) and the multi-level types and their relations investigated by the MLM community.

4.1 BORO Foundational Ontology background

The BORO Foundational Ontology [3], [52] is an extensional four-dimensional ontology. Its metaphysical architecture (choices) is described in [53]. It was developed in the late 1980s, originally for legacy reengineering enterprise systems [52]. It has recognised the importance of highlighting multi-level-types from the start. An early paper [54] focused on this topic and the book [3] has extensive discussions. It was not originally developed as a constructional ontology, though there is clearly a great deal of sympathy for constructivism. There is much talk of, for example, sets being constructed from their members in [3]. Furthermore, its extensional approach greatly simplifies the translation into a constructive framework. There has been some prior work on this translation over the last five years. It is mentioned briefly in [52], a more detailed description was given in [2] and was further developed in [55]. This prior work is further developed in this paper, in particular, the last paper introduces the coordinate system case we use in this paper.

4.2 Building a SIMPLE ontology in stages

We have all the apparatus in place to start to assemble ontologies for our four kinds of object. Here we take advantage of the framework's ontological space sandpit approach to build the target ontology a piece at a time. We start by defining our target ontology in terms of given and constructor domains. Then, in ontological space, we develop a series of sandpit ontologies that build up to the target ontology item by item. This will demonstrate how the assembly of the ontology is simple, largely a matter of choosing the configuration one wants. Though, of course, it requires substantial skill and knowledge to make an informed choice.

BORO's foundational structure is motivated by its metaphysical choices. These have been described many times in other papers (for example, [53]), so we do not cover this again here. Instead, when we look at the individual items, where appropriate we give a brief explanation of how we were influenced by the metaphysical choices.

4.3 Target ontology signature

We now look at the domains and later we build these up in ontological space.

4.3.1 Given domain

Our target's given domain contains a single mereological atom – an object of kind thing. This is the PluriVerse, a fusion of all possible things, which in BORO includes all possible worlds [45]; its acronym is 'PV'.

4.3.2 Basic constructors

Table 19 shows the three of the four kinds of objects in our object domain. We give the BORO variants their own names to clearly distinguish them.

Table 19 – Object domain – BORO kinds of object

Vanilla Name	Thing	Set	Sequence
BORO Variant Name	Elements	Types	Tuples

The corresponding three base constructors and their configurations (discussed above) are shown in Table 20. All construction is grounded.

Table 20 – SIMPLE's basic constructor domain

Kind	THING	SET	SEQUENCE
Constructor Type	THING-DECOMPOSER	SET-BUILDER	SEQUENCE-BUILDER
BORO Variant	ELEMENT-DIVIDER	TYPE-BUILDER	TUPLE-BUILDER
BORO Acronym	ED	TyB	TuB
CLAP Profile	CLAP (Orthodox)	C~~L~~AP (Orthodox)	~~CLAP~~ (Orthodox)
Empty Application	No	No	No
Stages	Population	Population	Population

Table 21 gives the BORO names of the base BORO Constructors' grounding relations.

Table 21 – Base BORO constructors' grounding relations

Vanilla Name	BORO Variant Name	BORO Acronym
set-member	types-instances	ti
thing-part	wholes-parts	wp
sequence-member-n	tuple-place-n	pn

In the exposition below, we typically first look at the shape that emerges from the construction ignoring the grounding relations and then subsequently consider the grounding relations.

4.3.3 Derived constructors

Table 22 lists SIMPLE's the two constructors in the derived constructor domain.

Table 22 – SIMPLE's derived constructor domain

Base Kind	Constructor Type	BORO Variant	Acronym
SET	SUBSET-BUILDER	SUBTYPE-BUILDER	STB
SET	POWERSET-BUILDER	POWERTYPE-BUILDER	PTB

Table 23 gives the BORO names of the derived constructors' grounding relations.

Table 23 – Derived BORO constructors' grounding relations

Vanilla Name	BORO Variant	Acronym
subset	super-sub-type	sst
powerset	powertype-instances	pti

4.4 *Ontological space*

Under our sandbox approach, we construct an ontological space with a series of ontologies that take us in small incremental steps from the NULL ontology to the target SIMPLE ontology. One could regard these as partial versions of the SIMPLE ontology, or subontologies of it. There are sixteen possible permutations of the target's given domain and individual constructors and Figure 22 shows the sixteen possible ontologies these generate the ontological space.

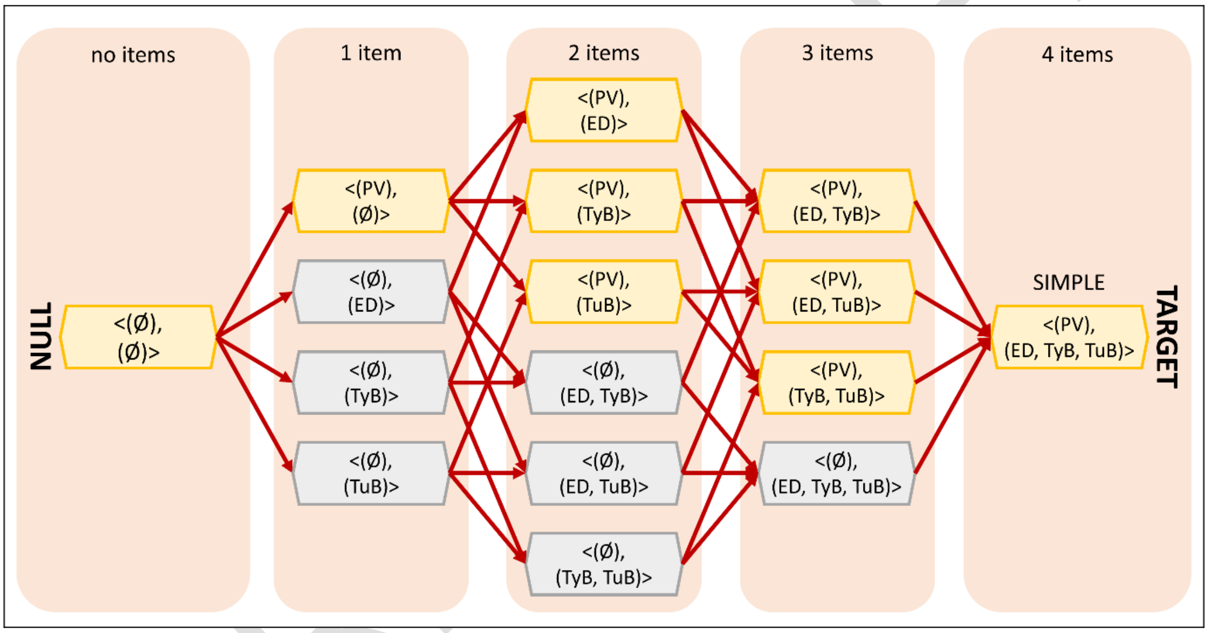


Figure 22 – Ontological space

Some of the ontologies with constructors do not generate any objects – they are greyed out in Figure 22 – and are not worth analysing. Table 24 lists the remaining ontologies, giving their names and identifying signatures.

Table 24 – SIMPLE's analysis ontological space

Ontology	Signature
NULL	<(Ø), (Ø)>
PluriVerse only	<(PV), (Ø)>
PluriVerse plus THING-DECOMPOSER	<(PV), (ED)>
PluriVerse plus TYPE-BUILDER	<(PV), (TyB)>
PluriVerse plus TUPLE-BUILDER	<(PV), (TuB)>
ELEMENT-DIVIDER and TYPE-BUILDER	<(Ø), (ED, TyB)>
ELEMENT-DIVIDER and TUPLE-BUILDER	<(Ø), (ED, TuB)>
TYPE-BUILDER and TUPLE-BUILDER	<(Ø), (TyB, TuB)>
PluriVerse plus ELEMENT-DIVIDER and TYPE-BUILDER	<(PV), (ED, TyB)>
ELEMENT-DIVIDER, TYPE-BUILDER and TUPLE-BUILDER	<(Ø), (ED, TyB, TuB)>
PluriVerse plus TYPE-BUILDER and TUPLE-BUILDER	<(PV), (TyB, TuB)>
SIMPLE	<(PV), (ED, TyB, TuB)>

We start the analysis with the NULL Ontology.

4.4.1 NULL: <(Ø), (Ø)> ontology

The NULL Ontology has no givens or constructors and so no objects [5, p. 278]. It is the root of the ontologies in the ontological space.

4.4.2 PluriVerse only (PVO): <(PV), (Ø)> ontology

We use this section to explain our choice of given. From a metaphysical viewpoint, the choice of givens typically involves important architectural commitments [53]. Typically, the givens are things, though this still leaves a variety of options. One could start with a domain of mereological atoms [56] and build up the ontology from them. Or one could adopt priority monism [28] (discussed earlier), then the given will be everything (which may be a single actual or a fusion of an infinity of possible worlds) and the ontology is built by decomposing this. As discussed earlier, whichever one of these is chosen would then dictate the direction of the THING constructor; whether to start with mereological parts and construct wholes – or vice versa, start with a mereological whole and construct the parts.

To keep things simple, we follow the priority monism route and have a given domain consisting of a single element (BORO's version of thing) – the pluriverse of BORO's possible worlds [45], which we will abbreviate as PV. Again, to keep things simple, we adopt super-substantivalism [31], [32], which Schaffer [21] calls monistic substantivalism; this considers matter to be identical to the spacetime region it occupies.

As there are no constructors in this ontology, its ontogenesis is the empty process. PV is the only element (and item) in the ontology. From a stage perspective, this is a limit case with a single stage 0.

4.4.3 PluriVerse plus TYPE-BUILDER: <(PV), (TyB)> ontology

With a given to provide input, the stages expand is a similar way to the SET-BUILDER ontologies we looked at earlier. Its ONTOGENESIS has this form:

```
STAGE [N]*
    (population [N-1])
        POWER (GROUNDED (TYPE-BUILDER))
        POWER (GROUNDED (SUBTYPE-BUILDER))
    (population [N-2]) POWER (GROUNDED (POWERTYPE-BUILDER))
    Where population [0] = PV
```

Like SET-BUILDER, this constructor's generative power is not exhausted as it progresses through the stages. To give an impression how it expands, we show in Table 25 the first three stages – ignoring derived constructors and grounding. Here one can clearly see that the generations expand non-linearly. Figure 23 is a visualisation of the three stages.

Table 25 –ONTOGENESIS <(PV), (TyB)> ignoring derived and grounding – first few stages

Stage	Input	Generated	Population
0			PV
1	PV	{PV}	PV, {PV}
2	PV, {PV}	{{PV}}, {PV, {PV}}	PV, {PV}, {{PV}}, {PV, {PV}}
3	PV, {PV}, {{PV}}, {PV, {PV}}	{{{PV}}}, {{PV, {PV}}}, {PV, {PV}, {{PV}}}, {{PV}, {{PV}}, {PV, {PV}}}, {PV, {{PV}}, {PV, {PV}}}, {PV, {PV}, {PV, {PV}}, {PV, {PV}, {{PV}}, {PV, {PV}}}	PV, {PV}, {{PV}}, {PV, {PV}}, {{{PV}}}, {{PV, {PV}}}, {PV, {PV}, {{PV}}}, {{PV}, {{PV}}, {PV, {PV}}}, {PV, {{PV}}, {PV, {PV}}}, {PV, {PV}, {PV, {PV}}, {PV, {PV}, {{PV}}, {PV, {PV}}}

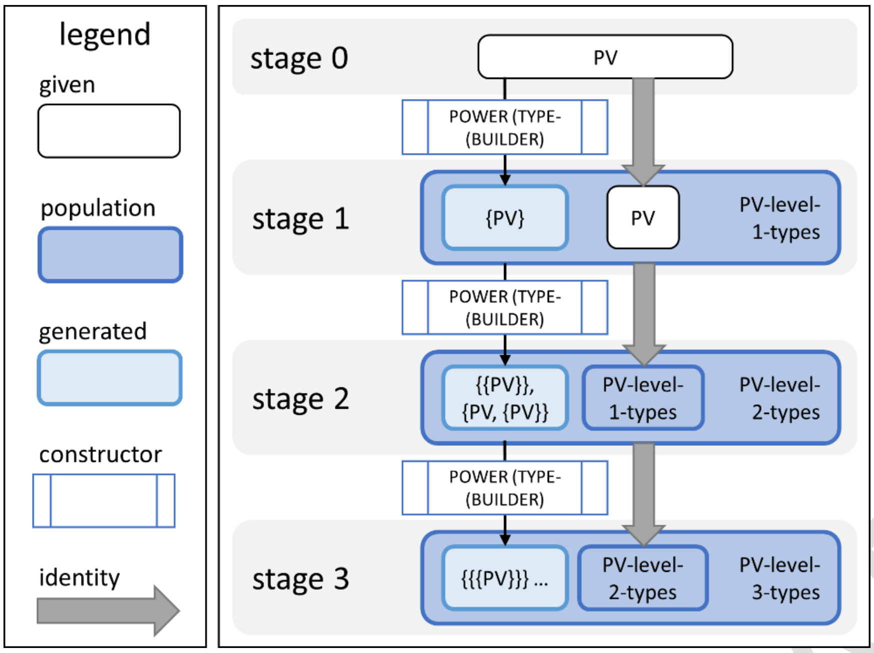


Figure 23 – ONTOGENESIS <(PV), (TyB)> – visualising stages 0 to 3 – population and generation

Recall TYPE-BUILDER does not accept an empty input, so there is no null set. As we are applying the whole population at each stage, the hierarchy is mixed. For example, the set {PV, {PV}} has members from both stage 0 and 1.

To give a rounded picture, we show the grounding relations generated from both base and derived constructors in the tables below. The type-instance grounding relations constructed in stages 1 and 2 are shown in Table 26. As the grounding relations are first class objects, they are included in the population for the next stage – we do not show that here.

Table 26 – Stages 1 and 2 type-instance grounding relations

Stage	Instance	Type
1	PV	{PV}
2	{PV}	{{PV}}
2	{PV}	{PV, {PV}}
2	PV	{PV, {PV}}

The derived POWER (SUBTYPE-BUILDER) does not generate super-sub-type grounding relations until stage 3, when it becomes prolific. A sample of these are shown in Table 27.

Table 27 – Sample of stage super-sub-type grounding relations

Stage	Sub-Type	Super-Type
3	{PV, {PV}}	{PV, {PV}, {{PV}}}
3	{{PV}, {{PV}}}	{PV, {PV}, {{PV}}}
3	{PV, {{PV}}}	{PV, {PV}, {{PV}}}
3	{{{PV}}, {PV, {PV}}},	{{PV}, {{PV}}, {PV, {PV}}}
3	{{PV}, {PV, {PV}}},	{{PV}, {{PV}}, {PV, {PV}}}
3	{{PV}, {{PV}}},	{{PV}, {{PV}}, {PV, {PV}}}

The derived POWER (POWERTYPE-BUILDER) is also slow to start, generating nothing in stages 1 and 2. The powertype-instance grounding relations generated in stages 3 and 4 are shown in Table 28.

Table 28 – Sample of stage powertype-instance grounding relations

Stage	Powered Type	Power-Type
3	{PV}	{{PV}}
3	{PV, {PV}}	
4	PV, {PV}, {{PV}}, {PV, {PV}}	{PV, {PV}, {{PV}}, {PV, {PV}}}

We can use the stage powertypes (discussed earlier) to visualise this ontology's stage structure – see Figure 24 (see also Figure 17).

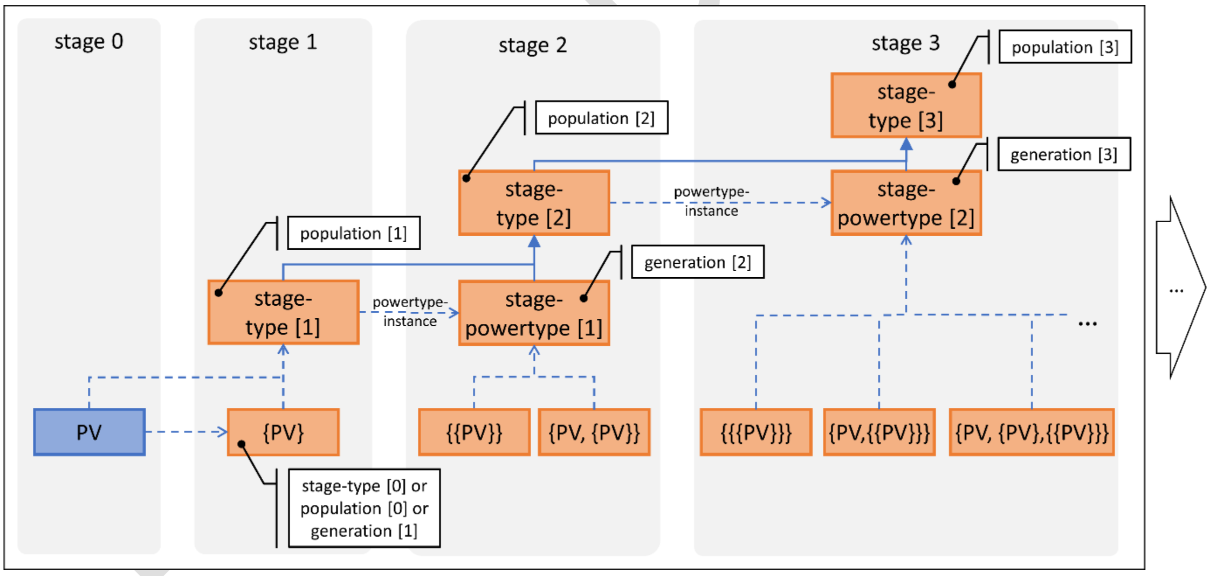


Figure 24 – Ontology's stage structure – powerset view

4.4.4 *PluriVerse plus ELEMENT-DIVIDER: <(PV), (ED)> ontology*

This ontology is an extension of the PluriVerse Only (PVO) Ontology above, extended with the constructor ELEMENT-DIVIDER. ELEMENT-DIVIDER works in the same way as THING-DIVIDER (see the earlier example THING-DIVIDER-EXAMPLE-1 ontology), taking a spatiotemporal element and dividing it into two parts.

The general stage proforma is:

STAGE [N]*

(population [N-1]) POWER (GROUND (ELEMENT-DIVIDER))

Where the givens - population 0 = (PV)

There is no point in including SUBTYPE-BUILDER or POWERTYPE-BUILDER without TYPE-BUILDER – as they will not construct anything. For symmetry, we could derive a corresponding POWER-PART-BUILDER but we do not as we do not need it here.

The general proforma does not make explicit what happens in the specific stages. At stage 1, the genie executes:

(PV) POWER (GROUND (ELEMENT-DIVIDER))

POWER here manages the non-deterministic repeated application of PV until the constructor is exhausted. This produces generation [1], all the possible parts of PV and exhausts the generative power of ELEMENT-DIVIDER. It also generates their grounding relations.

At stage 2, the genie executes:

(population [1]) POWER (GROUND (ELEMENT-DIVIDER))

Where (population [1]) = (PV + generation [1])

This does not generate any new objects, but the re-construction maps the remaining the possible whole-part grounding relations. It also exhausts the non-deterministic ELEMENT-DIVIDER – there are no possible new input and output combinations. Table 29 summarise this finite ONTOGENESIS (excluding grounding relations), where population 1 = PV + generation 1 = generation 2 = things. This is reflected in Figure 25.

Table 29 – Finite ONTOGENESIS

Stage	Input	Stage Operations	Generated	Population
0				PV
1	PV	POWER (ELEMENT-DIVIDER)	generation 1 (elements)	PV + generation 1 (elements)
2	population 1 (elements)	POWER (ELEMENT-DIVIDER)	generation 2 (elements)	population 1 + generation 2 (elements)

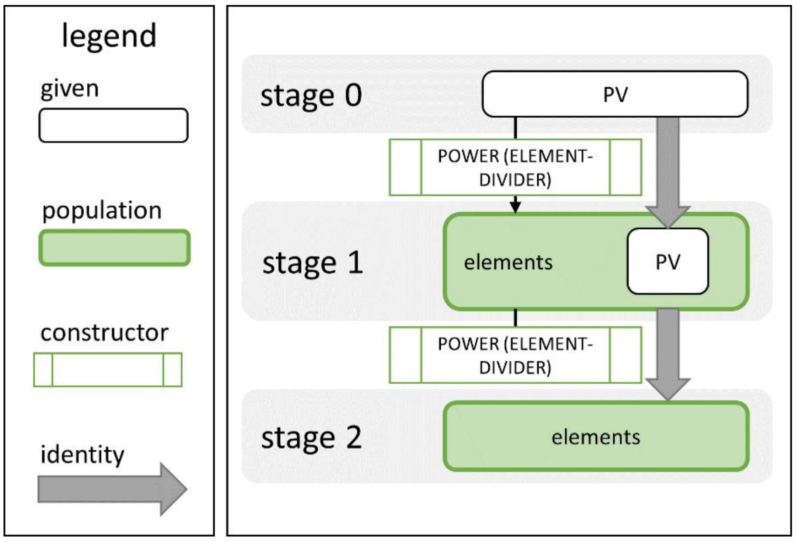


Figure 25 – ONTOGENESIS <(PV), (EB)> – visualising stages 0 to 2 – population and generation

4.4.5 PluriVerse plus TUPLE-BUILDER: <(PV), (TuB)> ontology

The stage proforma is:

STAGE [N]*

(population [N-1]) POWER (GROUND (TUPLE-BUILDER))

Where population 0 = PV

TUPLE-BUILDER (like SEQUENCE-BUILDER) is sensitive to order and repetition. So, the genie running its POWER operation applies all possible permutations of the input plurality in all possible orders and repetitions. As there is a single given, at stage 1 the sequences are all possible repetitions of this:

(PV<PV), (PV<PV<PV), and so on

Applying these constructs the corresponding sequences:

<PV, PV>, <PV, PV, PV>, and so on

At stage 2, the input plurality is (population [1] + generation [1]):

(PV, <PV, PV>, <PV, PV, PV>, and so on)

Generation [2] is sequences constructed from all possible permutations of this in all possible orders and repetitions. Here is a sample:

<<PV, PV>, <PV, PV, PV>> and <PV, <PV>>

This includes second level sequences, sequences with sequences as their components.

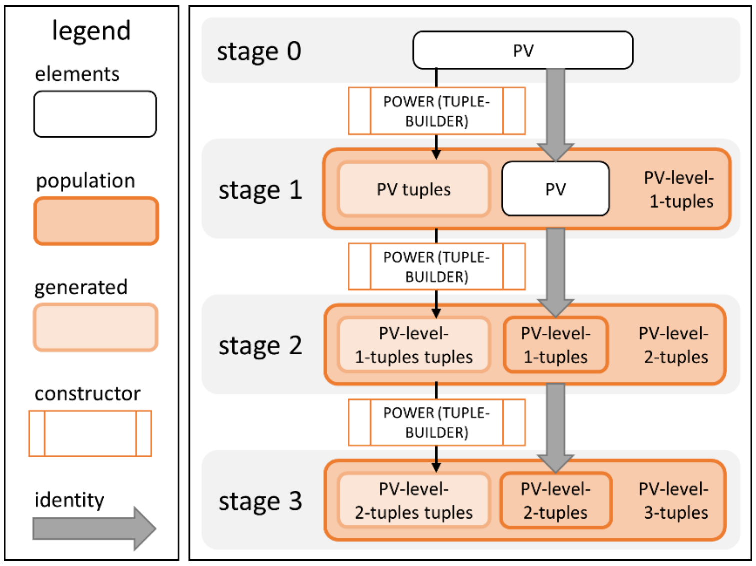


Figure 26 – ONTOGENESIS <(PV), (TuB)> – visualising stages 0 to 3 – population and generation

4.4.6 PluriVerse plus ELEMENT-DIVIDER and TYPE-BUILDER: <(PV), (ED, TyB)> ontology

As mentioned earlier, we adopt split staging for ELEMENT-DIVIDER when there are multiple constructors to keep the stages and levels matched. We know that ELEMENT-DIVIDER is exhausted in two stages, so we execute these two stages and then we start executing the TYPE-BUILDER and their associated derived constructors' stages.

STAGE [1]: (PV) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [2]: (population [1]) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [N]*

(population [N-1])

POWER (GROUND (TYPE-BUILDER))

POWER (GROUNDED (SUBTYPE-BUILDER))

(population [N-2]) (POWER (GROUNDED (POWERTYPE-BUILDER)))

Where N starts at 3.

4.4.7 PluriVerse plus ELEMENT-DIVIDER and TUPLE-BUILDER: <(PV), (ED, TuB)> ontology

This is much like the previous ontology with TUPLE-BUILDER replacing TYPE-BUILDER. The split stages have a similar format:

STAGE [1]: (PV) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [2]: (population [1]) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [N]*

(population [N-1]) (POWER (GROUND (TUPLE-BUILDER))

Where N starts at 3.

We could derive similar SUB-SEQUENCE and POWER-SEQUENCE constructors, keeping the symmetry even closer. But we have no need for these derived constructors here, and it is simpler to omit them.

We understand that it is unusual in conceptual modelling (and more specifically MLM) to consider tuple levels: mainstream multi-level modelling does not consider tuple as a basis for level hierarchies. However, if one considers tuples from a construction point of view, then this is a natural step.

4.4.8 The SIMPLE ontology

In the sandpit, one moves step by step building up the target ontology. We are ready to make the final step, merging the last two ontologies into the target ontology. As in both of those, we adopt split staging for ELEMENT-DIVIDER, but keep strict staging between TYPE-BUILDER and TUPLE-BUILDER. This gives us the following stage format:

```
STAGE [1]: (PV) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [2]: (population [1]) POWER (GROUND (ELEMENT-DIVIDER))

STAGE [N]*

    (population [N-1])

        POWER (GROUND (TYPE-BUILDER))

        POWER (GROUND (TUPLE-BUILDER))

        POWER (GROUNDED (SUBTYPE-BUILDER))

    (population [N-2]) POWER (GROUNDED (POWERTYPE-BUILDER))

    Where N starts at 3.
```

For our coordinate system case study, we found it useful to split the kinds – creating some internal structure. We follow the pattern set in the SPLITTING-SET-BUILDER example shown in Figure 15. We construct exhaust processes for TYPE-BUILDER and TUPLE-BUILDER; ELEMENT-DIVIDER already has one. We organise them much as we did in the example – giving us the similar structure in Figure 27. From this we get two hierarchy bases; things and tuples – and two levelled hierarchies – thing-types and tuple-types.

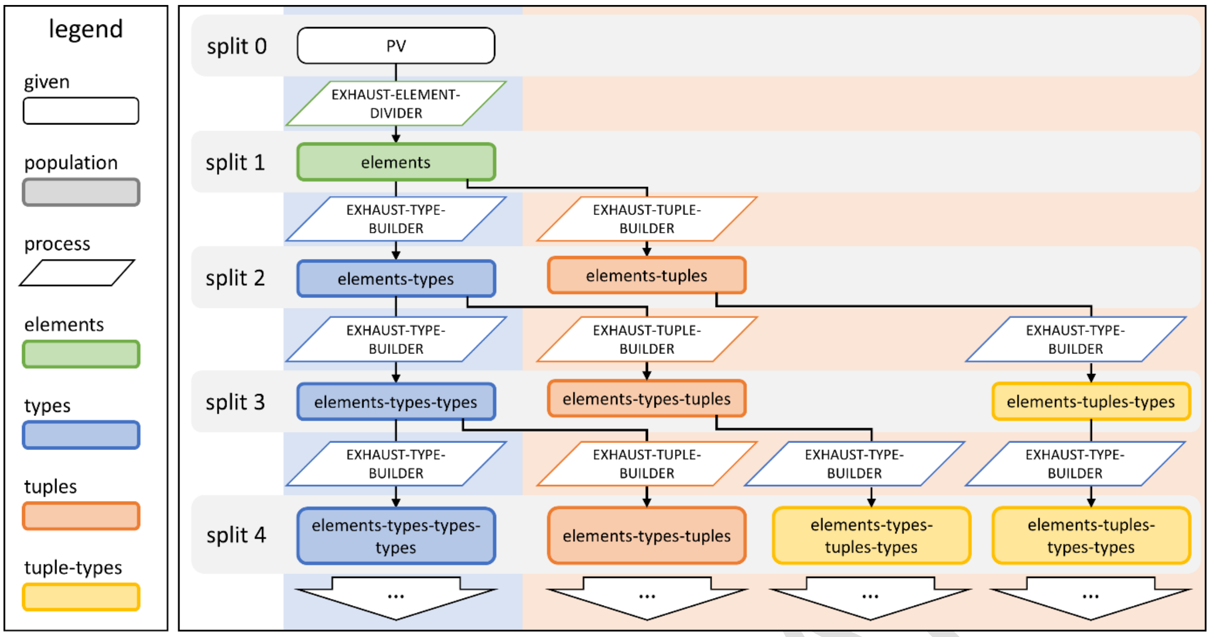


Figure 27 – Splitting SIMPLE into kind levels

4.5 Options (or ways of arranging) as levels

The project has started to use the SIMPLE ontology to characterise coordinate systems' options. We have found that the TYPE-BUILDER and TUPLE-BUILDER constructors and their thing-types and tuple-types levelled hierarchies (the split hierarchies fabricated above) give rise to two broad kinds of option which we have labelled; combination and permutation. We give simple, illustrative examples here to introduce these, to make it easier to understand the project-based examples described later. Though this association of levels with options may be uncommon in conceptual modelling, something similar is found in combinatorial mathematics, where combinations and permutations are well-known types of arrangements: hence these examples will look familiar to those found there.

4.5.1 Combination options example

Consider a game for two players that is played with just the four aces (the 'cards') from the standard deck of 52 playing cards. Assume that at the start of the game, someone deals these into two equal piles of two cards – two 'hands'. Let's say that hand one contains the aces of hearts and diamonds and hand two the other two aces, clubs and spades. We have talked about hands (piles), but what type of object could these be. An obvious candidate is the type (set) of the two cards – {hearts, diamonds}. If we want to talk about the way the cards have been dealt, the two hands (piles) that resulted – a 'deal' – then the obvious candidate is the set of the two sets – {{hearts, diamonds}, {clubs, spades}}. If we want to talk about all the possible ways the cards could have been dealt, all possible deals, then this would be the set of all three possible deals – shown in Figure 28 – the deal combinations.

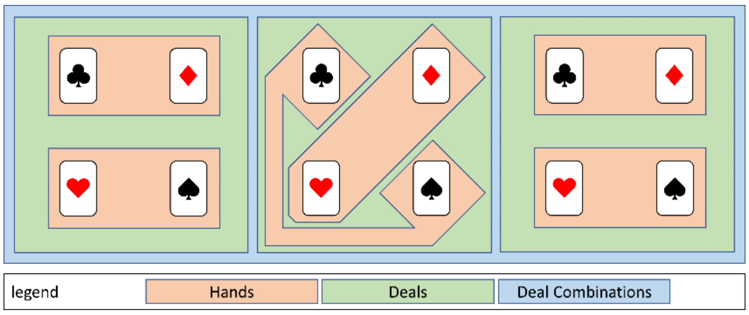


Figure 28 – Set of all possible deals – the three deal combinations

From a constructional point of view (and simplifying a little), we start with the four aces and then apply TYPE-BUILDER to all combinations of two cards to give us the possible hands. We then apply TYPE-BUILDER to all combinations of two hands that are disjoint, to give us all (three) possible deals. We then apply TYPE-BUILDER to all the deals – which gives us the single object 'deal combinations'. When hands are dealt, then the deal will be one of these. The construction is shown diagrammatically in Figure 29. Deal Combination is called a 'combination' (option) because it contains the different possible ways of combining hands.

The construction process makes clear how each object emerges from ascending a level in the ONTOGENESIS process. Figure 29 shows diagrammatically individual cards emerging in stage 1, individual hands in stage 2, deals in stage 3 and the deal combinations object in stage 4.

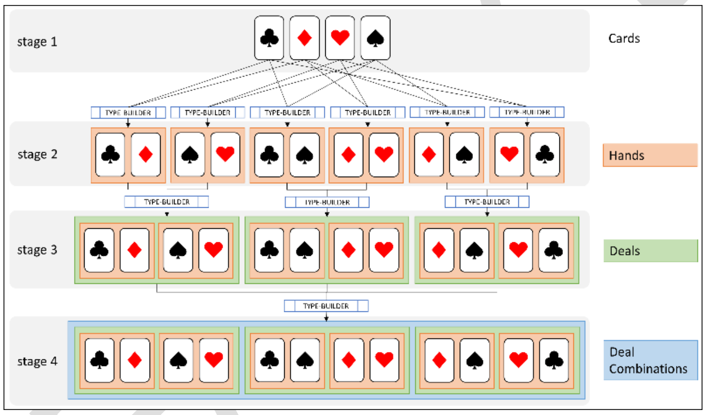


Figure 29 – Deal combination construction

4.5.2 Permutation options example

So far, we have not considered which player gets which hand. Reconsider the deal where hand one contains the aces of hearts and diamonds and hand two the other two aces, clubs and spades. What ways could these hands be distributed (permuted) across the two players – the player deals? Player A could have hand one and player B hand two – or vice versa. Clearly, there are two player deal permutation options, both with the same content but ordered in different ways – see Figure 30. These options are called permutation options because they are ways of arranging the hands among the players. And, as the members of a set are not ordered, sets of hands cannot capture this permutation structure, but tuples construction can; the tuple <hand one, hand two> is different from the tuple <hand two, hand one>.

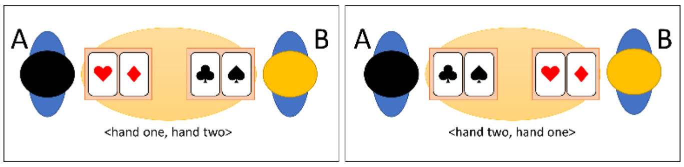


Figure 30 – Two player deal permutation options

We can construct these two players' deal permutations as tuples by taking the two instances of the deal type and applying TUPLE-BUILDER to the permutations of the two instances. More generally, we can create all the possible player deal permutations by taking every deal instance of the deal combinations object and then applying TUPLE-BUILDER to the two possible permutations of their two instances. If we apply TYPE-BUILDER to these tuple permutations, we get the object *player deal permutations* object.

As before, the construction process makes clear how the objects emerge from ascending a tuple level in the ONTOGENESIS process; in this case, tuple permutations. Individual *cards* emerge in stage 1, individual *hands* in stage 2, *player deals* in stage 3 (via POWER-TUPLE-BUILDER) and the *player deal permutations* in stage 4. This is shown diagrammatically in Figure 31.

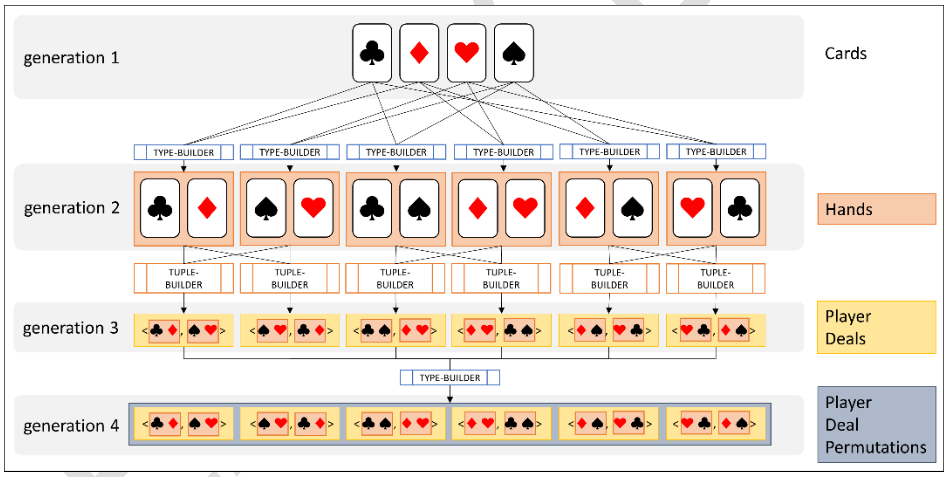


Figure 31 – Player deal permutations genesis

The constructional approach shows clearly how the individual permutations are generated by TUPLE-BUILDER – and so how permutations involve ascending tuple levels.

5 The coordinate systems' characterising options constructional ontology

In this section, we provide a brief simplified overview of the overall coordinate system constructional ontology as a context for the multi-level option examples in the next section. We hope to provide a fuller description of this ontology in future papers.

5.1 Analysis through geometric construction

The analysis is a kind of logical construction in the spirit of Bertrand Russell ("Wherever possible, substitute constructions out of known entities for inferences to unknown entities." [57, p. 363]) and Rudolf Carnap [58]. It involves a search for both the grounding geometric objects and how these are constructed. It usually takes substantial analysis and experimentation to identify suitable grounding components and associated elegant and parsimonious construction processes.

It became clear from early in the analysis that there were two sub-ontologies in play for this topic. The 'pure' coordinate system ontology and a deixis ontology that explains how the platform is, in practice, used to situate the coordinate systems. In the next two sub-sections, we give a brief overview of these two. In this short paper, we only have space to describe enough of the construction process to provide a background for the example options in the next section.

5.2 The deixis ontology – situating the coordinate system

In practice, the local coordinate systems are centred and oriented around the platforms and sensors that use them. These platforms often have their centre and orientation physically marked using a series of physical plates specified in the design. We call this the object's deixis or *attitude*; its orientation in space-time. Typically, this deixis is conceptualised as three orthogonal axes (geometrically, lines) [59] – shown in Figure 1. We label these lateral, longitudinal and sagittal, using the standard anatomical deixis terms [60]. In the ontology, we separate this concern into a discrete deixis sub-ontology. This is then used as a basis for orienting the coordinate systems.

5.3 The coordinate system ontology – options

Sets of co-oriented geometric surfaces are fundamental components of the ontology. In this section, we explain how these emerge and then look at the overall stages in the construction of the system.

The first two major options for a coordinate system are:

1. Reference frame – this is typically fixed by the object – the local reference frame.
2. Coordinate system's surface configuration type – in this case, one of the three surface configuration types in scope.

The three surface configuration types in scope are compositional: they each decompose into three reusable components – sets of co-oriented coordinate surfaces. What distinguishes the system types is the combination of different surface components, as enumerated in Table 30.

Table 30 – Coordinate system and their surface types

Coordinate System	Component Coordinate Surface Types
Cartesian	3 × planes
Spherical	sphere, cone and half-plane
Cylindrical	cylinder, half-plane and plane

The empirical evaluation of the different possible architectures against the data helped us to evolve a common, staged geometric construction process across the coordinate surface components – with variations for the different surfaces – as listed in Table 31.

Table 31 – Common staged coordinate surface construction process

Order	Stage	Description
1	Surface Orientation	Selecting the set of co-oriented surfaces
2	Solid Ordering	Building a mereological ordering for the surfaces – the process varies by surface.
3	Ratio Scaling	In these three systems, shifting down one or two dimensions to distance and angle ratios.
4	Unitising	Selecting the unitised distance or angle ratios – based upon the selection of unit.
5	Labelling	Labelling the unit ratios

As this shows, much of the work in constructing the overall system happens at the coordinate surface level. In this paper, we take most of our examples from the first two stages. The coordinate system is then assembled from its three surface components. The systems align the components, typically aligning their degenerate surface members, where these exist. The degenerate surfaces lose one or more dimensions, and so are lines or points – for example, the sphere with zero radius is a point. So, the intersection of three surfaces, one from each component set, picks out a point (as shown in Figure 32) – and conversely, every point can be picked out by the intersection of three surfaces. When each coordinate surface is given a coordinate numeral label, each point is named by the coordinate triple composed of the coordinate labels of the three surfaces that include it.

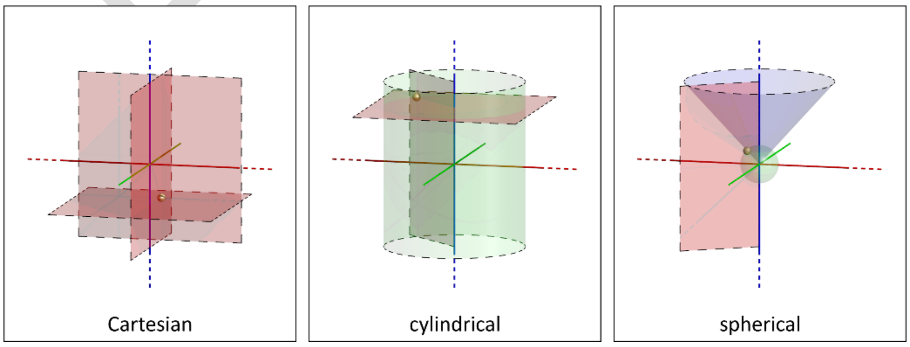


Figure 32 – Three coordinate systems – showing the intersecting surfaces that identify a point

6 Options as levels – examples

The previous sections have set up the context for the coordinate system examples of options as levels. The coordinate system ontology's broad stages contain a series of steps, which sometimes involve options. We have selected examples to illustrate the two types of options (combinations and permutations) and how these emerge from the constructional ontology.

6.1 Coordinate surfaces orientation combinations

As noted above, selecting the object selects the reference frame. And, selecting the type of coordinate system, picks out the types of the three surfaces that will be used. Each of the places in the coordinate triple contains a numeral that labels a surface of the chosen type.

To see what surfaces these are, consider the object at a point in time. Assume that the selected coordinate surface type is planes. Consider all the possible planes in its local reference frame. Partition these into sets of planes that are parallel to one another – the set of these is 'co-oriented plane types', a combination. Each of these subsets will contain planes that cover the whole of space; in other words, each point in space will be in one and only one plane in every subset. Also, all the subsets are disjoint, as members from different subsets will not be parallel – so cannot be in the same subset. A visualisation of this construction in Figure 33 (simplified using split THING-BUILDER and TYPE_BUILDER stages) shows how stage levels underpin the 'co-oriented plane types' combination.

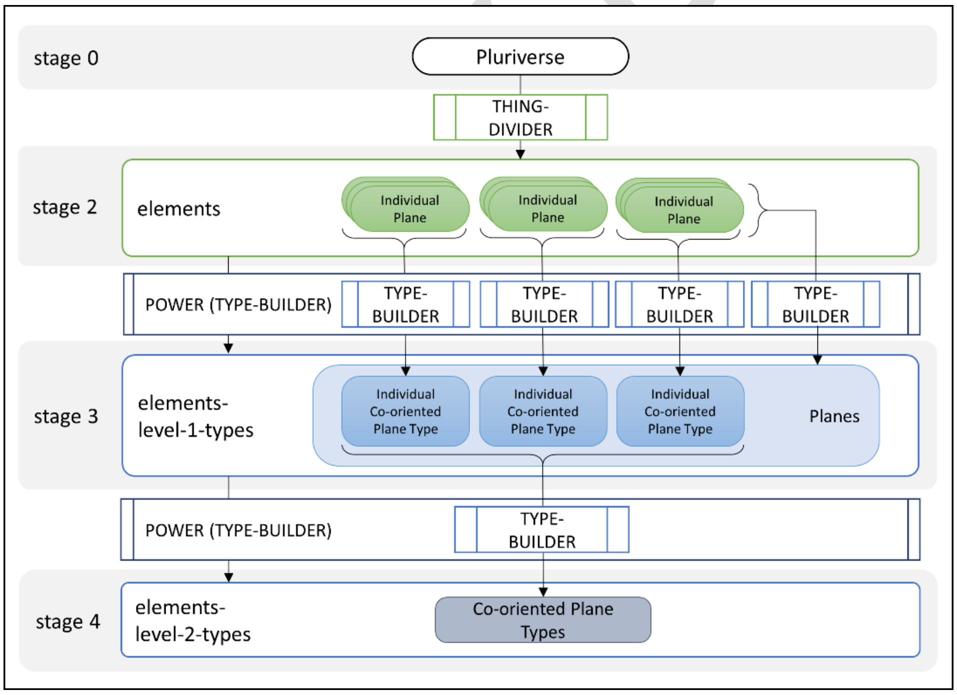


Figure 33 – Coordinate plane surface – generational levels

There is a similar geometric construction with variations for the other surfaces. For example, in the case of spheres, we start as before, by considering all the possible spheres in the objects' reference frame. We then partition these based on sharing a centre – in other words, being co-centred, hence co-oriented.

When setting up the coordinate surface for the coordinate system, the surface's orientation needs to be selected. In the case of planes, spheres and the other surfaces, the options for orientation are the instances of the relevant sets of co-oriented surfaces – which are level (generation) 3 objects – visualised in Figure 34. The three surfaces can then be grouped into a coordinate proto-system – a system with only orientation.

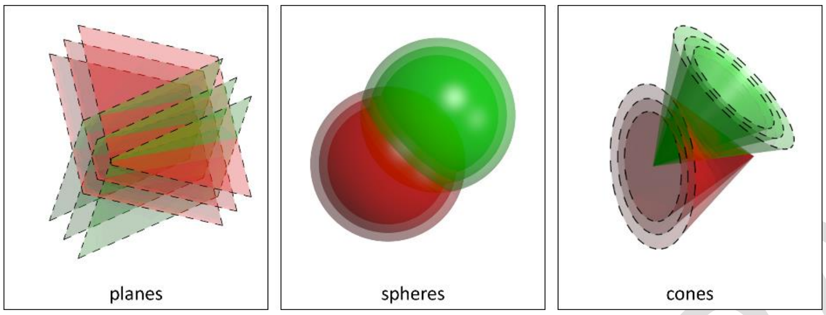


Figure 34 – Visualisation of sets of co-oriented surfaces – planes, spheres and cones

6.2 *Conical surface solidification combinations*

The goal of the solid ordering or solidification process in Table 31 is to end up with a set of solids – one for each surface – where the solids are mereologically linearly ordered – so, for every solid each of the other solids is either a part of it or has it as a part. This is needed for the next ratio scaling phase, which uses anthyphairesis – a mereologically based process of reciprocal subtraction; for details see [61, 62].

The analysis shows that the surfaces require different solidification constructions with different levels of options. It took some investigation to devise elegant and parsimonious constructions for some of the more complicated constructions. Spheres and cylinders are relatively simple with no pragmatic options; the solid is their finite interior (it would be unnatural to select their infinite exteriors).

Cones are a useful example of something with a less simple, but not too complicated option. Co-orientation partitions the set of cones (surfaces) into disjoint sets of co-oriented cones. It similarly partitions the set of conical solids into corresponding disjoint sets of co-oriented conical solids; in other words, each set of co-oriented cones has one corresponding set of co-oriented conical solids. One way of visualising this is to consider how each cone in a set of co-oriented cones, being infinite, divides the space into two half-spaces, both of which are in the corresponding set of co-oriented conical solids – giving a one-to-two mapping, see Figure 35.

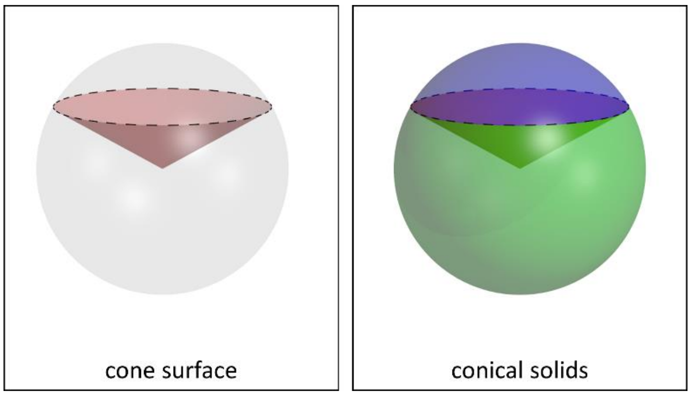


Figure 35 – Cone dividing space into two conic solids

This one-to-two mapping from surfaces to solids is a specific case of a more general situation that is reflected downstream in ambiguity of angle identification. In the simplest case, in plane geometry, where an angle is defined as a relationship between two rays meeting at a vertex, there is an ambiguity about which of the two angles is intended (in our case, it is which of the two solids) – see Figure 36. The rays by themselves are insufficient to distinguish between the two possible angles.

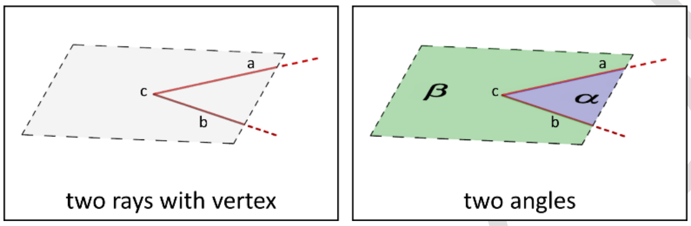


Figure 36 – Angle identification

This can be avoided if one defines 'angle' in a similar way to the Ancient Greek Carpus of Antioch, as a space between the lines – quoted in [63, pp. 125–126]. Then in Figure 36, there are two spaces, α and β, and therefore, two angles. Our solidification strategy moves up a dimension and takes the volume between two surfaces.

The Carpusian solid angle approach enables us to distinguish between the two angles. However, we also need a simple, consistent way to choose between the two conical solids (angles) associated with each cone in a set of co-oriented cones – which preserves a linear mereological ordering that is then reflected in the eventual labelling. We do this by exploiting the fact that each set of co-oriented conical solids contains two rays as degenerate solids: where the three-dimensional conical solids collapse into a one-dimensional ray (half-line) with no volume. These two solids are mereologically minimal; in other words, no other solid in the set is part of them. We then divide the solids sets into two subsets depending upon which of the degenerate solids they have as a part. We can use the degenerate solids as an index for each subset. Importantly, each subset is the basis of a different coordinate system component (hence a combination option), as it will result in a different way of labelling the solids (and so surfaces) with an angle. If we do not know which subset was chosen, we cannot interpret the label for the angle.

6.3 Coordinate surface – planar solid subsets permutations

The solidification process for planes is more complex, involving more options. As for cones, co-orientation partitions the set of planes (surfaces) into disjoint sets of co-oriented planes. In the case of planes, one constructs the set of parallel-boundaried planar solids – where one takes every pair of parallel planes (possibly identical) and constructs the solid that has the two planes as boundaries; in other words, the interior between the two planes (it would be unnatural to select either of the two infinite unbounded exteriors). Then co-orientation partitions this set into corresponding disjoint sets of co-oriented (parallel) planar solids; where each set of co-oriented planes has one corresponding set of co-oriented planar solids.

Note that every plane (surface) is contained in this parallel-boundaried planar solids type as a degenerate solid. Associated with each degenerate solid is a subset of this type containing every solid that has the degenerate one as a boundary. In coordinate system terms, these subsets represent the combination options for picking a reference plane, namely, the plane associated with the subset. The mereological structure of the solids in each subset induces two linear orderings each of which covers a half-space – the two orderings have the reference plane as their only common element. Intuitively, these orderings arrange mereologically the solids on each side of the reference plane.

The scaling-ratio-unit process will label the surfaces via the solids. But the labels will not be unique as things stand – they will appear twice, once in each subset. So, the subsets need to be differentiated (rather than one selected) and there are two ways (in the sense of permutations) of doing this, each way giving a different labelling and thus a different coordinate system. One can do this with a couple, where for clarity, we label place 1 as positive and place 2 as negative. Hence, this is a permutation option.

6.4 Deixis mapping to Cartesian proto co-ordinate system permutations

As noted at the beginning of this section, we extract the object's attitude (its orientation in space-time) from the coordinate system ontology into a deixis sub-ontology. This deixis is used to situate the coordinate system, and so the component coordinate surfaces. It turns out that the deixis axes are not fundamental to the pure coordinate system structure, and so are analysed away. However, they play a critical part in how the deixis situates the coordinate system. We illustrate this with an example that focuses on orientation, as this is both clearer and simpler than accounting for the full coordinate system.

We can define the orientation of a coordinate system as the set of orientations of its component surfaces. In our terms, the orientation of a coordinate surface is the set of co-oriented surfaces (this is analogous in some ways to Frege's abstraction from parallel lines to directions [64]). This notion of oriented surfaces can be easily extended to account for the orientation of a coordinate system – and regarded as a stage in its construction. An oriented coordinate system is one where its three surfaces have all been oriented; we call this a proto coordinate system.

As a side note, from this perspective, the Cartesian coordinate system's orientation does not involve axes directly – it is just a set of three co-oriented plane types. The conventional way of representing this with three orthonormal axes, one for each plane surface type, may be simpler to visualise but is misleading about the underlying ontology – as any of the infinite lines parallel to it will construct an identical orientation, the same set of co-oriented planes. One can regard the axes as the mechanisms for constructing the co-oriented surfaces rather than co-orientation itself.

However, the Cartesian axes are a good foundation for the deixis geometry – the object's attitude – as these are not arbitrary as they go through the object's centre. The three lines (axes without direction) are the deictic base orientation. However, some aspects of this orientation are abstracted away in the mapping to the Cartesian coordinate system. One way to appreciate this is to note that the same Cartesian proto system is constructed in the cases where the deictic axes are swapped (in other words, the axes are permuted). Physically, this would happen if an Unmanned Underwater Vehicle were to rotate in a way that any of its three deictic axes were swapped – as shown in Figure 37.

The six configurations in Figure 37 are the six ways that the situated object can view the Cartesian coordinate proto system. Another way to look at it is as the six ways to order the three surfaces or as six ways to interpret the three Cartesian coordinates relative to the object's attitude. Given three coordinate labels {x, y, z}, which is longitudinal (up-down) from the deictic perspective? Any one of the three could be. Hence the deictic base orientation has six permutations of the underlying Cartesian coordinate system orientation – in other words, this is a permutation option. For the full system, one would need to consider other factors, such as direction (for example, up versus down) which would multiply out the permutations.

In a single platform sensor system, there is less of a need to consider this point. There is less of a requirement to distinguish between the deixis and coordinate systems, and hence historically many single platform systems have merged the two.

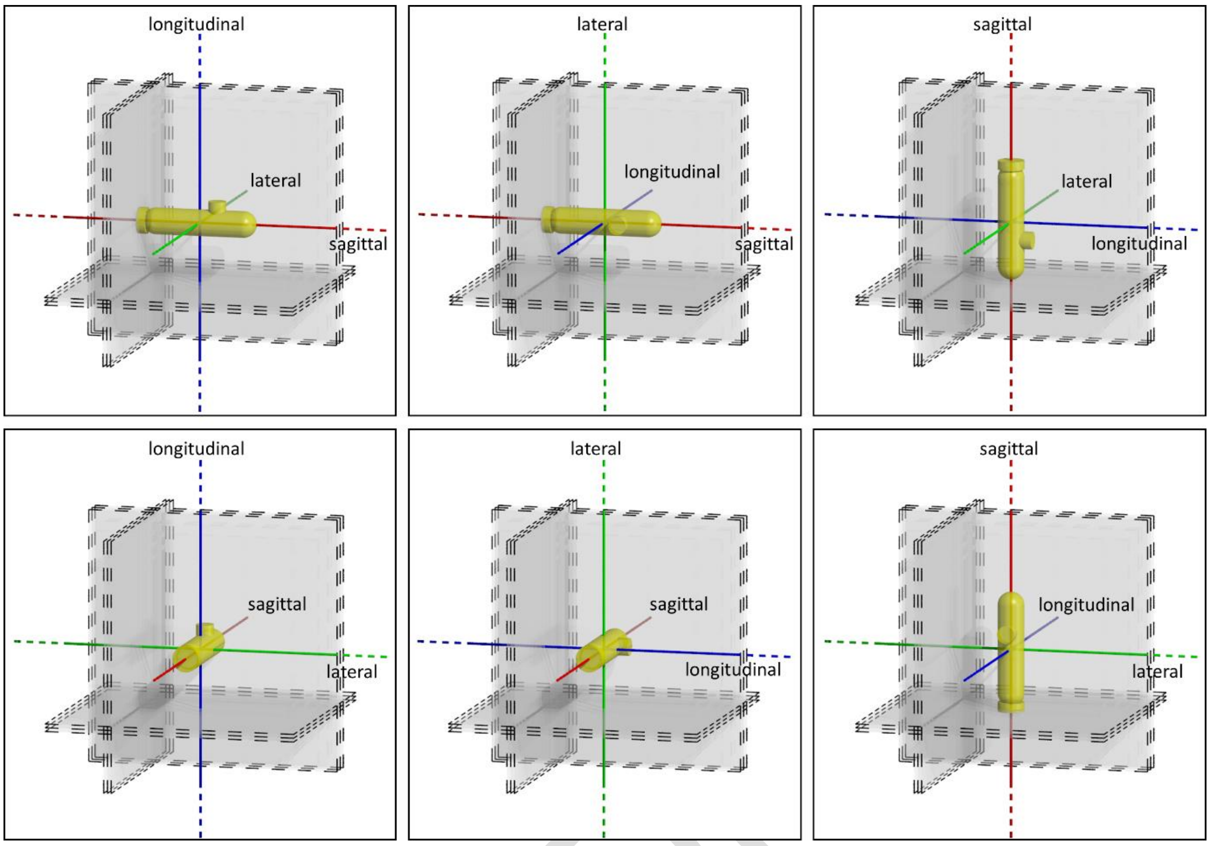


Figure 37 – Six permutations of the base deictic orientation

7 Conclusion

Despite the length and detail of the paper, we have exposed only a small part of what constructional ontology is capable of – just enough to illustrate how it applies to our coordinate system case study. Also hopefully enough to enable some readers to share Fine [6] and our vision of its power and beauty; as well as its ability to provide a single and elegant account of a variety of metaphysical structures.

7.1 Future work

The paper describes an approach to the analysis of the coordinate system domain – and introduces the framework developed to support the analysis. There is still a substantial amount of further detailed work on this project, only some of which is mentioned in the paper. Some of it is on the foundation; for example, the derivation of constructors to provide symmetry for SUBSET and POWERSET. Most of it is on the details of the domain, especially on the units of measure.

One can see the paper as implicitly making a case for using this approach on other domains. This would require the development of the detailed materials needed to support this kind of work.

More radically, we have a suspicion that there is a possibility for further scaling down. Fine's analysis of the formal identity principles [6] seem to help themselves to text's easy representation of linear order and repetition. This then leads to the need for the variety of pluralities shown in the taxonomy in

Figure 10. We wonder whether there is some way we can dissolve this complexity, letting it emerge from simpler constructions that build order and repetition, removing the need for exotic pluralities.

7.2 Summary

After spending some time building the framework, we were able to introduce two kinds of multi-level options; combination and permutation. With this framework, we were able to show how these options are ubiquitous in the domain analysis of coordinate system's characteristics – and how they can be explained using scaled down constructors that were built as part of the BORO constructional ontology. We have shown how the constructional approach gives a clear picture of how these options fundamentally involve ascending levels – and so are intrinsically multi-level. We have also shown, through the permutation examples, how the relatively unknown tuples/relations multi-levelling complements the more familiar type multi-levelling. There has not been much MLM work done in geometric domains; our unearthing of multi-levelling patterns in this domain should add weight to the claims (often made in the MLM community) that multi-level modelling pervades conceptual models in many domains.

Though it is not the main goal of the paper, we have provided some insight into how the conceptual foundation for a multi-platform sensed position coordinate system could be developed using the constructional approach and what it would look like. We have included some examples that show how using a foundational ontology can reveal that the underlying fundamental structure is not transparent. For example, how the analysis replaces the traditional visualisation of the Cartesian system as three axes with the less easy to visualise but more correct three co-oriented plane types. These help to make a case for a foundational analysis that can reveal the underlying structure in both this and other cases.

Constructional ontology is a relatively unexplored technical subject. So, as a precursor to assembling the SIMPLE BORO constructional ontology, we need to develop a general framework and approach as a toolkit for the assembly. This included how one can use an ontological sandbox to build up an understanding of the target ontology by stepping through ontological space. We also looked at how a liberal notion of composition based upon Finean CLAP principles could be used as the basis for a family of constructors to provide a rich and deep structure – rich and deep enough to characterise and analyse the core MLM modelling structures. We have shown how the constructional approach exposes the underlying compositional metaphysical structures of ontologies; for example, how SUBSET is derived from the fundamental SET-BUILDER constructor – implying generalisation is derived from instantiation. We have shown how understanding the CLAP principles of composition can expose the architectural choices made and suggest alternatives. We have shown how familiar domain hierarchical structures, such as taxonomies and component breakdowns, can be derived from more fundamental structures. We have shown how the same techniques can be applied to MLM classification and generalisation structures. We have provided a clear picture of the trade-off between order and expressiveness that would drive the choice of strict metamodelling (effectively applying generations rather than populations to the constructors).

Though this framework is by no means complete, we demonstrated how it provided enough structure for its elements to be simply configured into SIMPLE BORO ontology. We also gave some idea of the skill and knowledge required in the choice of elements and configurations.

While a lot more work remains to be done, we hope we have made a good case for using constructional ontology to understand to help us understand and improve the metaphysical grounding and other ontological underpinnings of conceptual models inherent in domain ontologies. We also hope that we have given you some idea of what we see as the power and beauty as well as the potential fruitfulness of this compositional constructional 'paradise'.

Acknowledgments: We want to acknowledge the help Steven Bradley, Salvatore Florio, Camilla Poulsen and Oscar Xiberta-Soto gave us in the development and preparation of this paper.

<u>REFERENCES</u>

1. Gardner, M.: The Fantastic Combinations of John Conway's New Solitaire Game "Life" . Scientific American, **223**, 120–3, (1970)
2. Partridge, C., de Cesare, S., Mitchell, A., Gailly, F., Khan, M.: Developing an Ontological Sandbox: Investigating Multi-Level Modelling's Possible Metaphysical Structures. MULTI-4th International Workshop on Multi-Level Modelling, 2017, pp. 226–34
3. Partridge, C.: Business objects: re-engineering for re-use. Butterworth-Heinemann 1996
4. Bohr, N.: On the constitution of atoms and molecules. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, **26**(153), 476–502, (1913)
5. Fine, K.: The Study of Ontology. Noûs, **25**(3), 263–94, (1991)
6. Fine, K.: Towards a Theory of Part. The Journal of Philosophy, **107**(11), 559–89, (2010)
7. Defense, U.S.D.: Unmanned Systems Integrated Roadmap FY2017 – 2042. 2018
8. Defense, U.S.D.: Unmanned Systems Integrated Roadmap FY2011 – 2036. Lulu.com 2015
9. Partridge, C., Lambert, M., Loneragan, M., Mitchell, A.: Semantic Modernisation: Layering, Harvesting and Interoperability. NATO Symposium IST-101 / RSY-024, Semantic and Domain-based Interoperability, 2011
10. Partridge, C., de Cesare, S., Mitchell, A., Odell, J.: Formalization of the classification pattern: survey of classification modeling in information systems engineering. Software & Systems Modeling, **17**(1), 167–203, (2018)
11. Partridge, C., Lambert, M., Loneragan, M., Mitchell, A., Garbacz, P.: A Novel Ontological Approach to Semantic Interoperability Between Legacy Air Defence Command and Control Systems. International Journal of Intelligent Defence Support Systems, **4**(3), 232–62, (2011)

12. Lambert, M., Partridge, C., Loneragan, M., Mitchell, A.: Demonstrating a Successful Strategy for Network Enabled Capability. NATO Symposium IST-101 / RSY-024, Semantic and Domain-based Interoperability, 2011
13. OMG: Open Architecture Radar Interface Standard (OARIS) v1.0. 2016
14. ISO: ISO/IEC 18026: 2009 – Information Technology – Spatial Reference Model (SRM). 2009
15. Suppes, P., Krantz, D., Luce, D., Tversky, A.: Foundations of Measurement, Vol. II: Geometrical, Threshold, and Probabilistic Representations. Academic Press 1989
16. Arntzenius, F., Dorr, C.: Calculus as Geometry. Space, Time, and Stuff, 2014, Oxford University Press (UK)
17. Grattan-Guinness, I.: Numbers, Magnitudes, Ratios, and Proportions in Euclid's Elements: How Did He Handle Them? Historia Mathematica, **23**(4), 355–75, (1996)
18. Partridge, C.: Geospatial and Temporal Reference – A Case Study Illustrating (Radical) Refactoring. ONTOBRAS-2013 6th Ontology Research Seminar in Brazil, 2013
19. Partridge, C.: An Information Model for Geospatial and Temporal Reference. 2011
20. Quine, W.V.: On What There Is. The Review of Metaphysics, 21–38, (1948)
21. Schaffer, J.: On What Grounds What. In: David Chalmers, David Manley, and Ryan Wasserman (ed.) Metametaphysics: new essays on the foundations of ontology, 2009, pp. 347–84, Oxford University Press
22. Seibt, J.: The "Umbau" – From Constitution Theory to Constructional Ontology. History of Philosophy Quarterly, **14**(3), 305–48, (1997)
23. Boolos, G.: The iterative conception of set. The Journal of Philosophy, **68**(8), 215–31, (1971)
24. Fine, K.: Our knowledge of mathematical objects. In: Tamar Szabo Gendler, John Hawthorne (ed.) Oxford Studies in Epistemology Volume 1, 2005, pp. 89 –109, Oxford: Clarendon Press
25. Fine, K.: The limits of abstraction. Clarendon Press 2002
26. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. The MIT Press 2000
27. Bennett, K.: Construction area (no hard hat required). Philosophical Studies, **154**(1), 79–104, (2011)
28. Schaffer, J.: Monism: The Priority of the Whole. Philosophical Review, **119**(1), 31–76, (2010)
29. Cantor, G.: Beiträge zur Begründung der transfiniten Mengenlehre. Mathematische Annalen, **46**(4), 481–512, (1895)
30. Sider, T.: Van Inwagen and the possibility of gunk. Analysis, **53**(4), 285–9, (1993)
31. Sklar, L.: Space, time, and spacetime. Univ of California Press 1977
32. Schaffer, J.: Spacetime the one substance. Philosophical studies, **145**(1), 131–48, (2009)
33. Linnebo, Ø.: Plural quantification. Stanford Encyclopedia of Philosophy, (2017)
34. Linnebo, Ø.: Pluralities and Sets. The Journal of Philosophy, **107**(3), 144–64, (2010)
35. Florio, S., Linnebo, Ø.: The Many and The One: A Philosophical Study. Forthcoming
36. Burgess, J.P., Rosen, G.: A subject with no object: Strategies for nominalistic interpretation of mathematics. Clarendon Press 1997
37. Florio, S., Nicolas, D.: Plural logic and sensitivity to order. Australasian Journal of Philosophy, **93**(3), 444–64, (2015)
38. Hewitt, S.: The Logic of Finite Order. Notre Dame Journal of Formal Logic, **53**(3), 297–318, (2012)
39. Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. ACM Transactions on Modeling and Computer Simulation (TOMACS), **12**(4), 290–321, (2002)
40. Kühne, T.: Matters of (meta-) modeling. Software and Systems Modeling, **5**(4), 369–85, (2006)

41. Russell, B.: The Principles of Mathematics. Cambridge University Press 1903
42. Zermelo, E.: Untersuchungen über die Grundlagen der Mengenlehre. I. Mathematische Annalen, **65**(2), 261–81, (1908)
43. Gödel, K.: The present situation in the foundations of mathematics. Collected works, 1933, pp. 45–53
44. Linnebo, Ø., Rayo, A.: Hierarchies ontological and ideological. Mind, **121**(482), 269–308, (2012)
45. Lewis, D.: On the plurality of worlds. Blackwell 1986
46. Lewis, D.: Parts of classes. Basil Blackwood 1991
47. Sanfilippo, E.M., Masolo, C., Borgo, S., Porello, D.: Features and Components in Product Models. Formal ontology in information systems: proceedings of the 9th International Conference (FOIS 2016), 2016, pp. 227–40, IOS Press
48. Frank, U.: Thoughts on classification/instantiation and generalisation/specialisation. 2012
49. Fine, K.: Neutral relations. The Philosophical Review, **109**(1), 1–33, (2000)
50. Chen, P.P.S.: The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems (TODS), **1**(1), 9–36, (1976)
51. Strachey, C.: Fundamental Concepts in Programming Languages. Higher-Order and Symbolic Computation, **13**(1-2), 11–49, (2000)
52. De Cesare, S., Partridge, C.: BORO as a Foundation to Enterprise Ontology. Journal of Information Systems, **30**(2), 83–112, (2016)
53. Partridge, C.: Note: A couple of meta-ontological choices for ontological architectures. LADSEB CNR, Padova, Italy, (2002)
54. Partridge, C.: Modelling the real world: Are classes abstractions or objects? Journal of Object-Oriented Programming, **7**(7), 39–45, (1994)
55. Partridge, C., Mitchell, A., Loneragan, M., Atkinson, H., de Cesare, S., Khan, M.: Coordinate Systems: Level Ascending Ontological Options. 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pp. 78–87
56. Goodman, N.: On relations that generate. Philosophical Studies, **9**(5-6), 65–6, (1958)
57. Russell, B.: Logical Atomism. In: J. H. Muirhead (ed.) Contemporary British Philosophy. Personal Statements, 1924, pp. 356–83, Allen & Unwin, London
58. Carnap, R.: The Logical Structure of the World and Pseudoproblems in Philosophy. University of California Press 1967
59. ISO: ISO 1503:2008 – Spatial Orientation and Direction of Movement – Ergonomic Requirements. 2008
60. Hyman, L.H.: Hyman's Comparative Vertebrate Anatomy. University of Chicago Press 1992
61. Fowler, D.H.: The Mathematics of Plato's Academy: A New Reconstruction. Clarendon Press 1987
62. Perry, Z.R.: Mereology and Metricality. Forthcoming in Philosophers' Imprint.
63. Proclus: A Commentary on the First Book of Euclid's Elements. Princeton University Press 1970
64. Frege, G.: The Foundations of Arithmetic: A Logico-Mathematical Enquiry Into the Concept of Number. Basil Blackwell & Mott 1950