

B usiness

O bject

R eference

O ntology

Program

Working Paper

M01

METHODOLOGY: OVERVIEW

**THE BORO APPROACH TO RE-
ENGINEERING ONTOLOGIES**

s
i
m
p
l
i
f
y
i
n
g

s
e
m
a
n
t
i
c
s

Copyright Notice © Copyright The BORO Program, 1996-2001.

Notice of Rights All rights reserved. You may view, print or download this document for evaluation purposes only, provided you also retain all copyright and other proprietary notices. You may not, however, distribute, modify, transmit, reuse, report, or use the contents of this Site for public or commercial purposes without the owner's written permission.

Note that any product, process or technology described in the contents is not licensed under this copyright.

For information on getting permission for other uses, please get in touch with contact@BOROProgram.org.

Notice of liability We believe that we are providing you with quality information, but we make no claims, promises or guarantees about the accuracy, completeness, or adequacy of the information contained in this document. Or, more formally:

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Contact For queries regarding this document, or the BORO Program in general, please use the following email address:

contact@BOROProgram.org



THE BORO APPROACH TO RE-ENGINEERING ONTOLOGIES

1	Introduction	M01-1
2	The BORO Methodology	M01-2
2.1	Picking an element of an existing computer system	M01-3
2.2	The worked examples' business patterns	M01-3
2.3	Entity formats based on working computer systems	M01-5
3	A systematic approach to re-engineering	M01-5
3.1	Re-engineering data	M01-5
3.2	Stages in the method	M01-6
3.3	Following the systematic method	M01-9
4	A framework for the model	M01-9
4.1	The framework level	M01-10
4.2	Other levels in the model	M01-10
4.3	Assuming that all classes are application level objects	M01-11
4.4	Levels as objects	M01-13



CONTENTS

M01

4.5 Expanding the framework level—the general lexicon	---	M01-13
5 Generalisation and compacting	-----	M01-16
5.1 Spotting that objects share the same patterns	-----	M01-16
5.2 Compacting metrics	-----	M01-16
6 The rest of the Worked Example Papers	-----	M01-18
BORO Working Papers - Bibliography	-----	M01-19

CONTENTS

M01





CONTENTS

M01



M O 1

M E T H O D O L O G Y : O V E R V I E W

THE BORO APPROACH TO RE-ENGINEERING ONTOLOGIES

1 Introduction

This paper is an introduction to the papers in the [MW—The BORO Methodology: Worked Examples](#) series. It gives an overview of the methodology BORO applies when re-engineering existing computer application systems. The rest of the papers describe the details of the methodology and how it is applied using worked examples - providing a good introduction to the methodology.

It usually makes sense to use a reference ontology as the starting point for the re-engineering - and to incorporate the results of the re-engineering in the reference ontology. But neither of these are essential.

This paper and the [MW—The BORO Methodology: Worked Examples](#) papers presuppose some familiarity with:

- the Entity Paradigm Ontology,
- the Object Paradigm Ontology, and
- the BORO Notation.



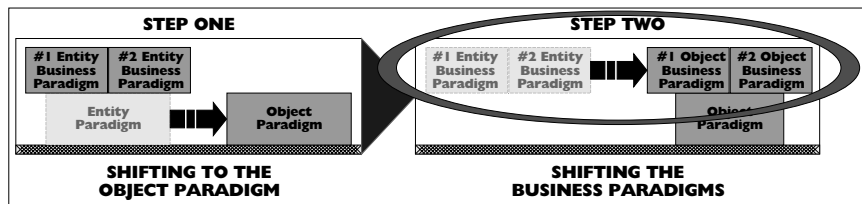
Comprehensive explanations of the Entity and Object Paradigm ontologies can be found in BORO's *O—ONTOLOGY Papers*. The details of the BORO Notation are described in BORO's *BG—Business Ontology: Graphical Notation*.

2 The BORO Methodology

The BORO methodology is designed to re-engineer the entity-based business ontology paradigms embedded in existing computer application systems—ones that are used for such tasks as accounting or foreign exchange trading. This re-engineering is what provides with the practical benefits of re-engineering.

The process of re-engineering the entity ontology business paradigms embedded in computer systems into object ontology business paradigms is illustrated in [Figure M01-1](#). We start by transforming the computer system's entity-formats into an object model, using the object ontology paradigm as a foundation. This re-engineering, by its very nature is structured. This has certain advantages. Because we are working within a framework, we can guarantee a practical result. This takes the uncertainty out of re-engineering.

Figure M01-1
Re-engineering
entity business
paradigms



Using entity-based computer systems as the starting point for our re-engineering enables us to define a systematic method for re-engineering. This makes the re-engineering both effective and efficient. With this methodology, the re-engineering of the entity ontology business paradigms can be planned and executed with a reasonable amount of precision. This makes it a viable commercial approach.



2.1 Picking an element of an existing computer system

In each example, we pick a small element of an existing computer system; typically:

- An entity type,

And, its associated;

- Attribute types,
- Entities, and
- Attributes.

We use these as the starting point for the re-engineering process. For each of them, we work through a series of steps that transmutes their entity formats into a more sophisticated object model.

The worked examples serve two purposes:

- First, they help us to understand what re-engineering a business ontology paradigm involves and how the methodology works.
- Second, they provide us with basic object models, which we can re-use again and again in subsequent re-engineerings. Essentially the beginnings of a Reference Ontology.

Because the examples include fundamental business patterns (ones that are used in a wide range of businesses), they are likely to be reusable in most re-engineerings and so provide a good kernel for the Reference Ontology.

2.2 The worked examples' business patterns

The Worked Examples Papers are organised into groups based on two types of business pattern:

- Spatial patterns, and
- Temporal patterns,



The BORO Approach to Re-Engineering Ontologies

2 The BORO Methodology

The first group—Spatial Patterns—is based on three simple entity formats that are found in many business computer systems:

- Country,
- Region, and
- Address.

And are described in the following three papers:

- MW-1 - Re-Engineering Country,
- MW-2 - Re-Engineering Region, and
- MW-3 - Re-Engineering Address.

Each re-engineering builds upon the previous one - and the final outcome is a comprehensive object model of our spatial patterns. In the process, we also re-engineer a general object model for naming patterns.

The second group—Temporal Patterns—is based on two simple entity formats:

- Bank holiday, and
- Weekend.

And these are described in the following papers:

- MW-4 - Re-Engineering Bank Holiday, and
- MW-5 - Re-Engineering Weekend.

As with spatial patterns, each re-engineering builds upon previous ones. The final result of this part of the re-engineering is a general object model of our temporal patterns.

The early examples are designed to help you become familiar with the systematic re-engineering method by carefully taking you through each step in the process. We begin with a simple example—country—whose re-engineering is straightforward. This provides a comfortable atmosphere in which to start establishing the basic principles of the methodology.



2.3 Entity formats based on working computer systems

The later examples, building on the established principles, are designed to help you understand what is happening in the re-engineering and to develop a feel for how entities are transformed into objects. You will learn how to construct simple general objects that, through re-use, compact the object model. And, you will begin to appreciate how accurate your analysis needs to be within the object ontology paradigm.

2.3 Entity formats based on working computer systems

All the entity formats in the examples have come from working computer systems. Some details have been changed, for reasons such as confidentiality and ease of understanding. They are common formats that could have come from any number of systems. In the examples, we assume that they come from a single working computer system (this is not too far from the truth).

3 A systematic approach to re-engineering

A systematic approach to re-engineering entity formats is essentially a replay, at a more specific level, of the re-engineering of the entity ontology paradigm into the object ontology paradigm. Having this overall pattern¹ greatly simplifies the methodology - and enables a formalisation into a step-by-step system described in the worked example papers.

3.1 Re-engineering data

The method focuses on re-engineering data—in particular the entity formats in an existing legacy system. In BORO's experience the data in most systems contains enough of the important business patterns to enable us to construct the general patterns needed for a comprehensive business model.

1. Described in the paper [OP1 - Entity Ontology Paradigm](#), and also in [Business Objects](#), Chapter 3.



The BORO Approach to Re-Engineering Ontologies

3 A systematic approach to re-engineering

Process is deliberately excluded by the method. Experience has shown that it is practically impossible to effectively re-engineer process into a business model. The problem is that process has a language-like linear structure. (That is why people talk about programming *languages* in contrast to database management systems.) Like everyday language (and unlike paper tables and computer data), these linear constraints so seriously distort process's view of the business that it cannot be re-engineered systematically. It greatly simplifies the re-engineering to exclude process. As mentioned above, this did not compromise the final model, as the more explicitly structured data contained more than enough of the object patterns we needed (including event patterns).

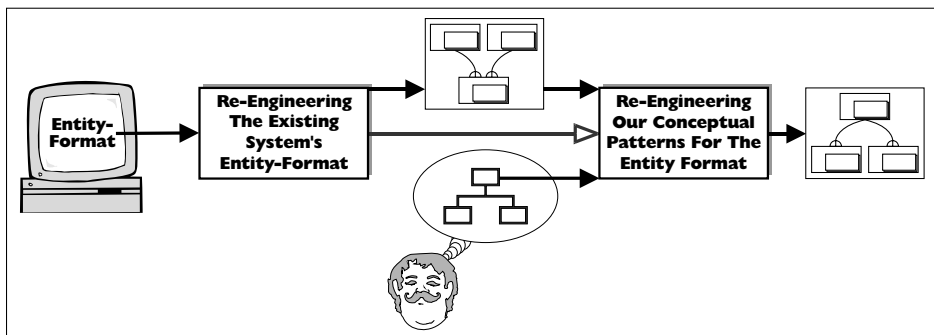
3.2 Stages in the method

The worked examples make more sense if we first look briefly at an outline of the methodology. This divides the re-engineering into two main stages:

- Re-engineering the entity format of the existing entity system, and
- Re-engineering our conceptual patterns for the entity format.

These are illustrated in [Figure M01-2](#).

Figure M01-2
Two stages of re-engineering

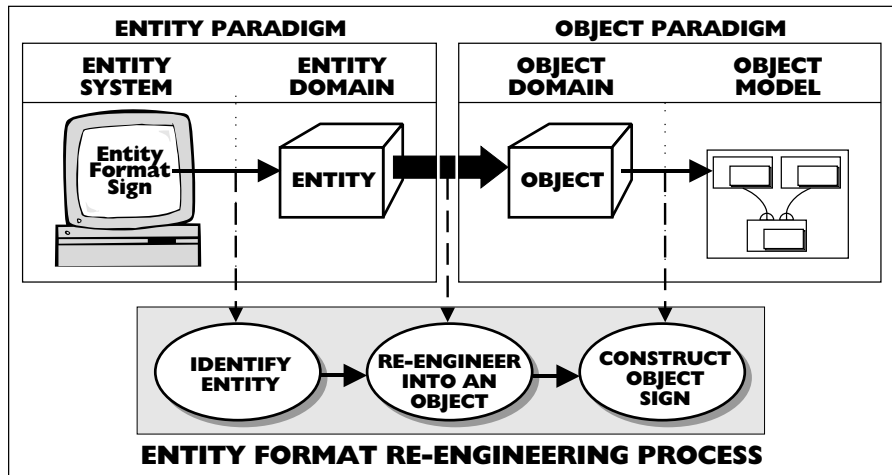


3.2.1 Re-engineering the entity format of the existing entity system

The elements of the entity format are re-engineered one by one. Each re-engineering starts with a sign in the entity format; from it an entity in the 'real world'

is identified. This entity is then re-engineered into an object and a sign constructed for the object in the model. This process is illustrated in [Figure M01-3](#).

Figure M01-3
Re-engineering
the existing
system's entity
format



Rules for
ordering the
elements of
the re-
engineering

We have found that it is worth following two simple rules when choosing the order in which to re-engineer the various elements of the entity formats. This makes the whole process of re-engineering much more straightforward.

The first rule is:

Re-engineer the individual entity and entity type signs before their associated individual attribute and attribute type signs.

This is only common sense. We obviously need to work on the entity sign, before we can move onto its dependant attribute signs.

The second rule—for individual entities and entity types—is:

Re-engineer a couple of individual entity signs and use the patterns to re-engineer their entity type sign.

This again is common sense. It is much easier to establish the patterns with more tangible individual entities than with 'abstract' entity types. This is sometimes called working by example.



The second rule only needs a change of name to apply to attributes. After the change, it reads:

Re-engineer a couple of individual attribute signs and use the patterns to re-engineer their attribute type sign.

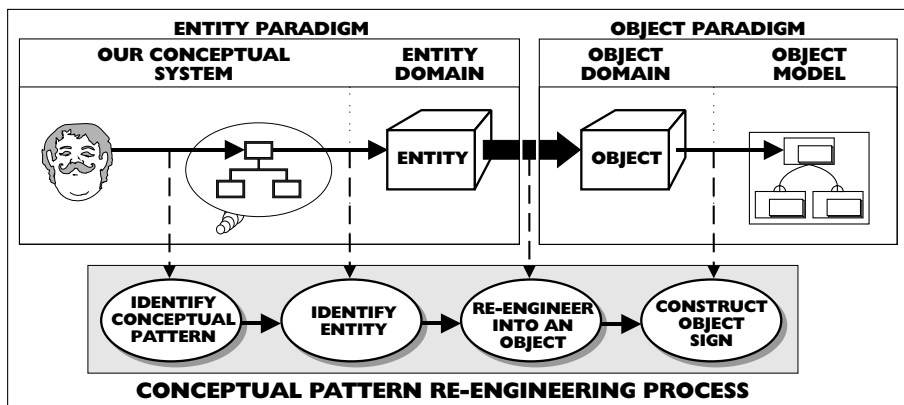
3.2.2 Re-engineering our conceptual patterns for the entity format

In the second stage, of the process we re-engineer our conceptual patterns (in other words, the patterns our brain uses) for the entities in the entity format. This follows similar steps to the first stage, with the addition of one initial step. We:

- Find a relevant conceptual pattern,
- Identify the entity (entities) the conceptual pattern refers to,
- Re-engineer it (them) into objects, and
- Then construct signs for the objects in the model.

These steps are illustrated in [Figure M01-4](#). The reason we need the extra first step (of finding a relevant conceptual pattern) at this stage is that—unlike the entity formats in the existing computer systems, these are not formally listed. It involves some effort, and in some cases, ingenuity to find relevant ones.

Figure M01-4
Re-engineering our conceptual patterns





3.3 Following the systematic method

In the worked examples, these two stages (and the steps within them) occur repeatedly. However, once we have firmly established the steps in the process, we focus on the nature of the re-engineering. This means that, in later examples, we do not work through each step in detail.

This should not be seen as a licence to miss out steps when you re-engineer. It is tempting to do so, but you should resist. Go through every step in the process, especially the first few times you re-engineer. Even someone who is experienced can easily take a wrong turning. Relying on intuition or a gut feeling, particularly at the start, is a sure recipe for slipping back into the old (entity) way of seeing and almost bound to lead you astray. Following the full process helps keep you on the straight and narrow.

Furthermore, and perhaps as important, when the model is being reviewed, your detailed workings will make it much easier for the reviewer to see any wrong turnings you may have taken. With the workings revealed, the re-engineering will not look like a series of magic rabbits appearing from nowhere out of top hats.

4 A framework for the model

In the worked examples, you will see that the objects have been allocated to one of these three broad levels:

- Framework level,
- Application level, and
- Operational level.

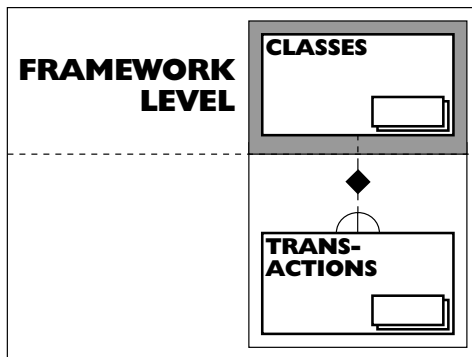
We have found that this not only makes the model easier to understand but helps in the system building process.

4.1 The framework level

The framework level contains the object meta-model. We have found it helps to have this explicitly described in the model. So, rather than start each project with a blank sheet of paper, We start with a framework meta-model.

In the object schemas, We identify the framework level with a background shading. [Figure M01-5](#) has a sample. Any model object within the shading is a framework object.

Figure M01-5
An example of
framework level
shading



4.2 Other levels in the model

There are two non-framework levels:

- The application level, and
- The operational level.

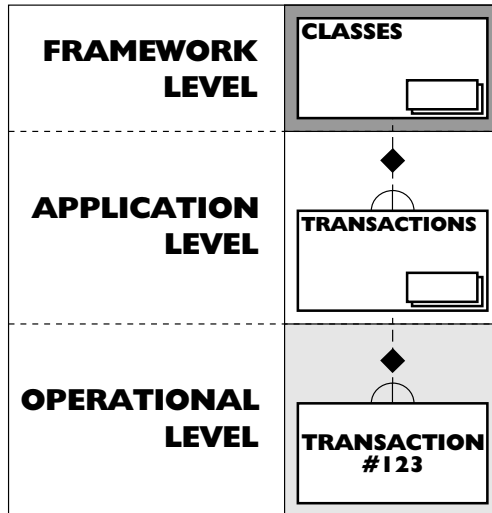
We call those model objects that will typically be constructed during the system development, application level objects. We call those that are typically set up during live operation by the users of the finished system, operational objects.

We use a similar shading method of identification to differentiate these two levels of objects. We shade the background of the operational level objects; so, by default, the application level objects have no background shading. [Figure M01-6](#) shows an example of the different shadings. In it the class transactions is in an

4.3 Assuming that all classes are application level objects

area with no shading signifying it is application level; whereas. transaction #123 is in an area of light shading signifying that it is an operational level. As you can see in the figure, the framework shading is darker than the operational shading.

Figure M01-6
An example of
operational level
shading



We find it useful to start modelling with individual objects, particular examples of the more general classes. These are more tangible and help me (and others) to see the underlying patterns more clearly. These individual objects are often operational level, objects that the users would set up. Consequently, we usually end up with quite a number of operational objects in our working models. When we tidy up the model for the next stage of system building, we purge them because they are not needed.

4.3 Assuming that all classes are application level objects

Working in the entity ontology paradigm, it is easy to assume that the application-operational and type-individual distinctions are the same or similar. In other words, that all entity types are application level and all individual entities are operational level. This assumption is wrong.



The BORO Approach to Re-Engineering Ontologies

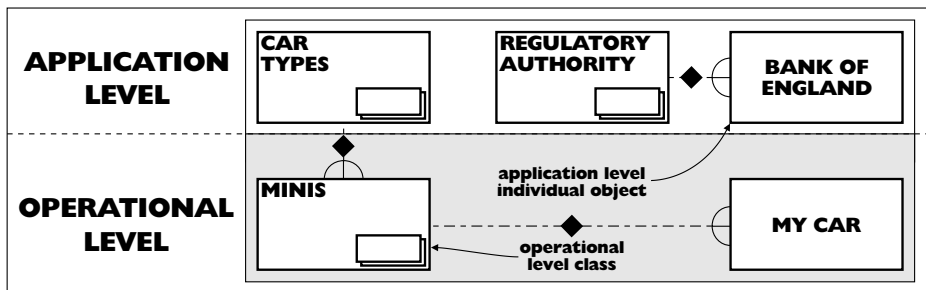
4 A framework for the model

Some people carry this mistaken assumption across into object modelling. They assume that all classes are application level and all individual objects are operational level. This places an unnecessary restriction on their modelling. We can show this with examples of classes that are at the operational level and individual objects at the application level.

Consider a system that keeps information about types of cars². Assume that one of the records on its Car Types file is named 'Minis'. If we work out what this refers to, it is the class of Minis. If we assume all classes are application level, we would classify the Minis class as application level. However, if someone was building a general package for car manufacturers, it would make no sense for them to construct in the system 'individual' car types, such as Minis, that vary from manufacturer to manufacturer. They would let each manufacturer set up their own. So Minis, despite being a class, is an operational object.

Now consider a general Bank of England reporting system for banks. We would expect this to contain information about the Bank of England as standard. Because it is an individual object, not a class, we might be tempted to allocate it to the operational level of our business model. Yet, it is not something that the users at individual banks would be expected to set up. So, despite being an individual object, it is an application level object. This and the last example are modelled in [Figure M01-7](#).

Figure M01-7
Examples of
application and
operational level
objects

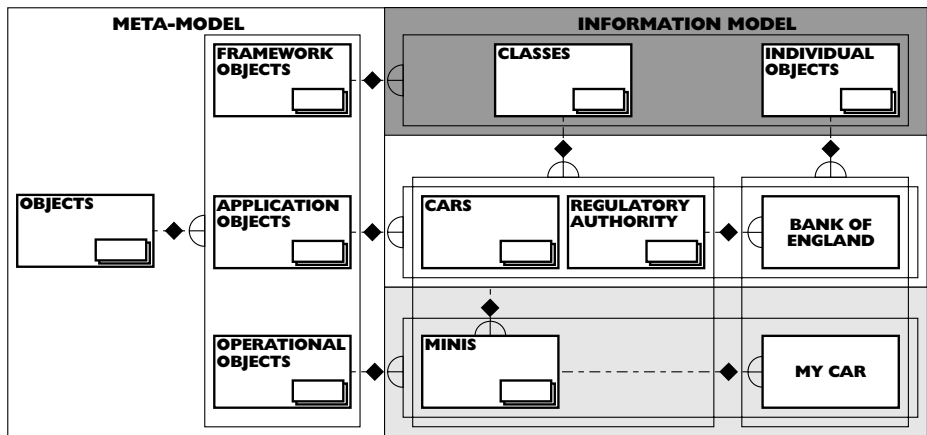


2. A system of this type is described in OP3 - Logical Ontology Paradigm - and modelled in its [Figures OP3-.28](#) and [OP3-29](#) (it is also in [Business Objects](#), Chapter 6 - and modelled in its [Figures 6.8](#) and [6.9](#)).

4.4 Levels as objects

In the object ontology paradigm, with its strong reference principle³, the level shadings are signs referring to objects—level objects. The framework level object is the meta-class of all the framework objects. Each level object is the meta-class of all objects in that level. These meta-classes are modelled in *Figure M01-8*, which also shows the individual application and overlapping operational objects illustrated in *Figure M01-7*—though without their connections.

Figure M01-8
Level objects



4.5 Expanding the framework level—the general lexicon

We find it useful to expand the framework level into a ‘starter pack’ for future re-engineering projects. Initially we call this a general lexicon, but when it reaches a suitable size, we call it a Reference Ontology.

4.5.1 What is a general lexicon?

In everyday language, a lexicon is a store of words or concepts. Dictionaries can be seen as, and in some cases, have been called lexicons. As far as a business object

3. The strong reference principle stipulates that every sign should refer to a corresponding object - and each object should have no more than one sign.



modelling is concerned, the general lexicon is a store of general re-usable model objects.

4.5.2 What is a reference ontology?

We have found that when the general lexicon reaches a certain size, it starts making managing, generalising and identifying similar objects across projects significantly more effective. At this stage it makes sense to start formally including as a basis for future re-engineerings - to mark this change in status we start calling it a reference ontology. It stores the model objects that have fruitful patterns, ones that are re-used in most models. We classify all these as framework level. So, for all practical purposes, the reference ontology is the framework level.

Sometimes we have found, and you will find, it useful to create specialist reference ontologies for specialist areas of the business. For instance, if you were to work on a number of accounting systems, you might construct an accounting reference ontology. This would contain models of the accounting objects that you found were re-used in most of the accounting models.

Reference ontology as a transformation of the Aristotelian categories

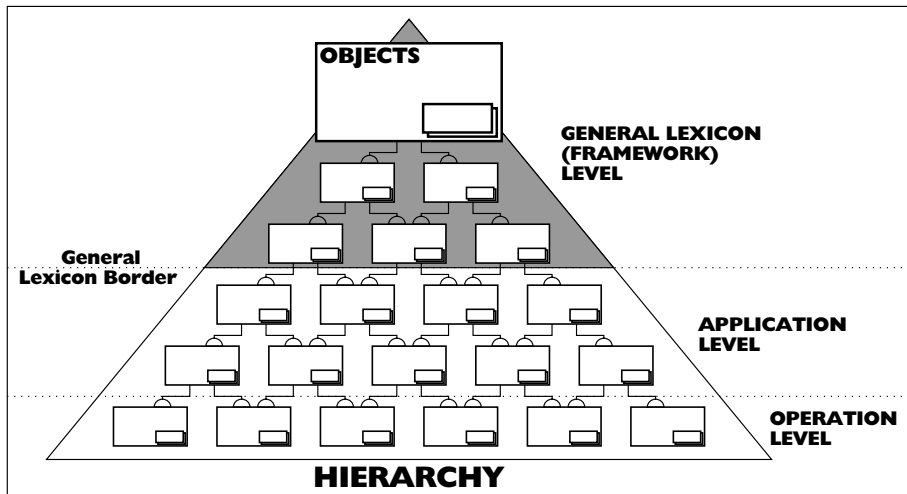
The reference ontology can usefully be seen as the object ontology paradigm's transformation of the Aristotelian categories (discussed in the paper [OP2 - Substance Ontology Paradigm](#) and illustrated in [Figure OP2—14](#)). The object meta-model embedded in the framework level is the transformed version of Aristotle's framework of categories—substance and types of attribute (quality, quantity, etc.). The rest of the reference ontology is the transformed version of the lower level Aristotelian categories. It records the same types of things, such as the humans class being a sub-class of the more general animals class.

4.5.3 Constructing a reference ontology.

An essential precursor to the construction of a useful reference ontology is identifying model objects that will be re-used. If, over a sufficiently large number of models, the model of a particular object (or group of objects) is re-used frequently, then it is a likely candidate. The naming pattern that appears in all of the first group of worked examples is a good example.

With experience, modellers find that they can judge whether model objects are candidates. There is a useful rule of thumb that helps them make that decision. It is the more general a model object, the more likely it is to be re-used; the less general, the less likely it is to be re-used⁴. Here, we measure generality in terms of how high up either the class-member or super-sub-class hierarchies the object is. The class objects is the limiting case; it is at the top of both hierarchies. As we move down the hierarchies from objects, we get closer and closer to the border of the framework and application levels - the framework border (illustrated in [Figure M01-9](#)).

Figure M01-9
The framework
border



The exact location of the framework border is a matter of opinion. Different people sometimes have slightly different views. In the end, it does not matter exactly where the border is, so long as the majority of really re-usable model objects are inside it, ready for re-use.

When we decide a model object belongs to the framework level, implementing this decision is trivial; we classify it as framework level. Because the reference ontology is tied into the framework level, whenever we classify a model object as

4. But it is only a rule of thumb. Some lower level objects (e.g. human) are better candidates for re-use than higher level objects (e.g. primate).



framework level, it automatically belongs to the general reference ontology. A finer grained system is needed, for specialist reference ontologies.

5 Generalisation and compacting

A key objective of business object modelling is generalising objects. This is what delivers compacting and its associated benefits.

5.1 Spotting that objects share the same patterns

An important element in generalisation is spotting that objects share the same patterns. The schemas are a useful tool for this. They describe the pattern of relations between objects. This is what Frege called the sense element of meaning—distinguishing it from the reference element⁵. The object schemas (and so the business model) map Fregean sense patterns. Mapping these sense patterns creates an environment that encourages generalisation. The object schemas play a key role. They can make the sense patterns explicit in a way that makes similar patterns easier to spot.

The way in which the schema is drawn influences how it is understood. In well drawn schemas, similar patterns have similar shapes, making them easier to spot. However, a badly drawn schema can make two similar patterns look dissimilar. So, it is worth spending some time drawing an object schema properly. We find that we sometimes go through five or more drafts before we arrive at something we am happy with. This is worth doing because it encourages substantial generalisation, and so substantial compacting.

5.2 Compacting metrics

We find that we can get a rough idea on how successful the re-engineering is by monitoring the scale of generalisation and compacting.

5. Frege's notion is described in many places, including the paper OP3 - Logical Ontology Paradigm and Chapter 5 of *Business Objects*.

5.2.1 Population, re-use and generalisation metrics

We usually measure the compacting directly with population metrics that compare the number of items in the existing entity system with the number in the object model. We also measure the actual amount of re-use and generalisation that has occurred in the re-engineering of the business model. By comparing the two sets of metrics, we can see the correlation between generalisation, re-use and compacting.

5.2.2 Model levels

We normally divide the metrics into levels. We do this for both the entity system and object models' metrics. The entity system's metrics are divided into the following two levels:

- Type, and
- Individual.

The object model's metrics are divided into the following three levels:

- Framework,
- Application, and
- Operational.

Candidate framework objects are treated as application objects until the end of the re-engineering. So the original framework level objects are separated and excluded from the overall metrics. The framework model is regarded as separate from the model for the particular re-engineering.

The application and operational level objects are included in the overall metrics, but a distinction is made between the two because only the application level objects are passed onto the systems analysis stage. By contrast, operational objects are eventually purged from the business model. These are just examples of the types of objects that the user will construct after the system has gone live. So these two levels can be used to separate the impact on compacting of



generalisation during development—the application level—from that during operation—the operational level.

Even though we formally purge the operational objects from the business model, we often use them during the later stages of system building. For example we might use them for illustrative examples during the systems analysis and also as the basis for the test data used in system testing.

6 The rest of the Worked Example Papers

This overview has introduced you to the major concepts involved in the BORO Methodology. The rest of the *MW—The BORO Methodology: Worked Examples*-worked example papers describe the details of how the process of applying them - and in so doing provide us with the beginnings of a reference ontologies.



BORO Working Papers - Bibliography

The BORO Working Papers

Volume A

A—The BORO Approach

Book AS

AS—The BORO Approach: Strategy

AS1—*An Overview of the Strategy*

AS2—*Using Objects to Reflect the Business Accurately*

AS3—*What and How we Re-engineer*

AS4—*Focusing on the Things in the Business*

Volume - O

O—ONTOLOGY Papers

Book - OP

OP—Ontology: Paradigms

OP1—*Entity Ontology Paradigm*

OP2—*Substance Ontology Paradigm*

OP3—*Logical Ontology Paradigm*

OP4—*Business Object Ontology Paradigm*

Volume - B

B—Business Ontology

Book - BO

BO—Business Ontology: Overview

BO1—*Business Ontology - Some Core Concepts*

Book - BG

BG—Business Ontology: Graphical Notation Constructing Signs for Business Objects



BORO Working Papers - Bibliography

Graphical Notation I

BG1— *Constructing Signs for Business Objects*

Graphical Notation II

BG2— *Constructing Signs for Business Objects' Patterns*

Volume - M

M—The BORO Re-Engineering Methodology

Book - MO

MO—The BORO Re-Engineering Methodology: Overview

M01— *The BORO Approach to Re-Engineering Ontologies*

Book - MW

MW—The BORO Methodology: Worked Examples

Worked Example 1

MW1— *Re-Engineering Country*

Worked Example 2

MW2— *Re-Engineering Region*

Worked Example 3

MW3— *Re-Engineering Bank Address*

Worked Example 4

MW4— *Re-Engineering Time*

Book - MA

MA—The BORO Re-Engineering Methodology: Applications

MA1— *Starting a Re-Engineering Project*

MA2— *Using Business Objects to Re-engineer the Business*

Book - MC

MC—The BORO Re-Engineering Methodology: Case Histories

Case History 1

MC1— *What is Pump Facility PF101?*



THE BORO APPROACH TO RE-ENGINEERING ONTOLOGIES

A-M

A

application level (of model) --M01-9–M01-12, M01-17
 classes ----- M01-11
 Aristotelian categories -----M01-14

B

business paradigm
 entity – re-engineering ----- M01-2

C

compacting
 metrics for -----M01-16, M01-18

D

distorted
 process's view of the business ----- M01-6

E

entity
 re-engineering order rules ----- M01-7
 entity paradigm
 entity-based computer systems ---- M01-2

re-engineering ----- M01-5
 explicit
 business model -----M01-10
 data ----- M01-6
 sense patterns -----M01-16

F

framework border -----M01-15
 framework level (of model) M01-9–M01-10, M01-13–
 M01-15, M01-17
 meta-model -----M01-10
 Frege, Gottlob -----M01-16
 fruitful patterns
 general lexicon -----M01-14

G

general lexicon -----M01-13
 Aristotelian categories -----M01-14
 generalisation
 compacting -----M01-16
 metrics for -----M01-16, M01-18

M

model levels (framework, etc.) ----- M01-9



O

operational level (of model) -M01-9-M01-13, M01-17

R

re-engineer

data and process M01-5

rules for ordering elements M01-7

systematic approachM01-5-M01-9

Reference ontology M01-13-M01-14

re-use

general lexiconM01-14

metrics for M01-17-M01-18

REV-ENG method

stagesM01-6-M01-8

S

sense and reference M01-16

strong reference principle

applyingM01-13