Business Object Reference Ontology Program

simplifying semantics

Working Paper

MW2

Methodology: Worked Example - 2

Re-Engineering Region

Issue: Version - 4.01 - 01-July-2001

Contact

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

contact@BOROProgram.org

MW2

METHODOLOGY: WORKED EXAMPLE - 2

RE-ENGINEERING REGION

CONTENTS

1 Introduction ---------- MW2-1

2 Re-engineering the region entity formats of the existing system -- MW2-2
- 2.1 Re-engineering the region entity type sign ---------- MW2-2
- 2.2 Re-engineering region entity signs ---------- MW2-3
- 2.3 Re-engineering the entity type sign ---------- MW2-5
- 2.4 Re-engineering region full name attribute type sign ---------- MW2-5
- 2.5 Re-engineering region code attribute type sign ---------- MW2-8
- 2.6 Generalising within region ---------- MW2-9
- 2.7 Generalising across region and country ---------- MW2-12
- 2.8 Re-engineering the region entity formats ---------- MW2-19

3 Re-engineering our conceptual patterns for region ---------- MW2-20
- 3.1 Nested regions ---------- MW2-21
- 3.2 Region stages ---------- MW2-26

4 Summary ---------- MW2-29

BORO Working Papers - Bibliography ---------- MW2-31

INDEX ---------- MW2-33

CONTENTS
MW2

BORO

MW2

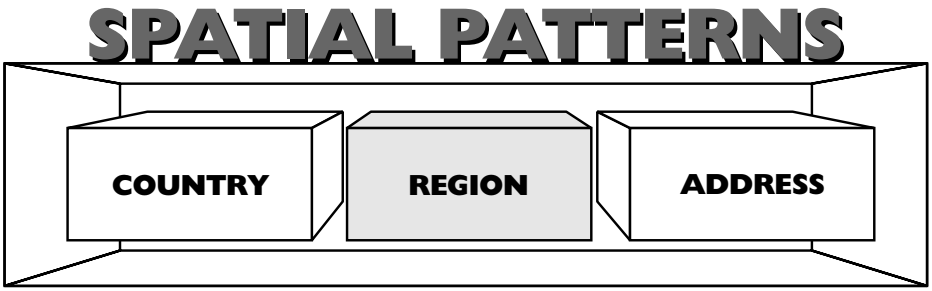
Methodology: Worked Example - 2

Re-Engineering Region

1 Introduction

In the last worked example (*MW1—Re-Engineering Country*), we re-engineered our first example of a spatial pattern, country. In this paper, we re-engineer our second example, region (see *Figure MW2–1*). We currently have the beginnings of an object model for spatial patterns. Re-engineering region enhances this. In the next paper (*MW3— Re-Engineering Bank Address*), we re-engineer the third and final example, address, which completes the model.

Figure MW2–1 Region, second of three examples of spatial patterns



Because we are now familiar with the patterns for the systematic re-engineering process, in this and future examples, we do not work our way through every step.

This not only helps us to take an overall view, it means we move quickly and avoid monotony. With region, we follow the systematic two-stage process for re-engineering. We:

- Re-engineer the region entity formats of the existing system, and
- Re-engineer our conceptual patterns for region.

2 Re-engineering the region entity formats of the existing system

Region is, in many respects, similar to country; so, we re-use its pattern of re-engineering. This means that there is potentially an opportunity to generalise across country and region, which we will look into later.

As with the country re-engineering, we follow the standard rules for re-engineering the entity formats:

- Re-engineer the individual entity and entity type signs before their associated individual attribute and attribute type signs.
- Re-engineer a couple of individual entity (attribute) signs and use the patterns to re-engineer their entity-(attribute)-type sign.

We start with the region entity signs, and then move onto their attribute signs: region full name and region code.

2.1 Re-engineering the region entity type sign

Like before, we familiarise ourselves with the entity type by looking at a list of its individual entities, such as the partial listing of regions in *Table MW2-1*. As with country, we are only interested in the two name attributes shown in the listing,

so we ignore region's other attributes. We can deduce from *Table MW2–1* that the part of the region format we are interested in looks like *Table MW2–2*.

Table MW2–1: Partial region listing

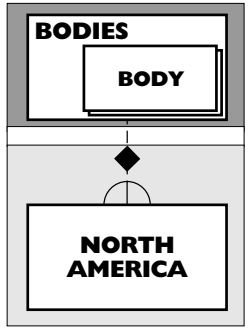
Region Name	Region Code
Europe	EU
Far East	FE
Middle East	ME
North America	NA

Table MW2–2: Region entity format

Entity type	Attribute type #1	Attribute type #2	Etc.
Region	Region full name	Region code	-

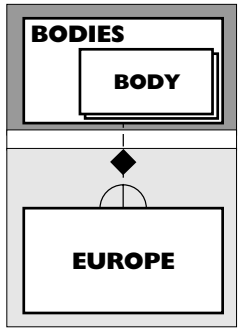
2.2 Re-engineering region entity signs

Figure MW2–2 North America object schema



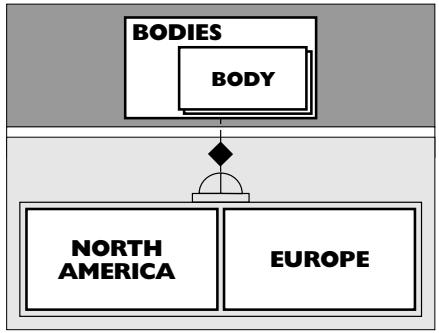
We follow the rules and start by re-engineering a couple of entities for the entity type. We need to start with an entity; so, we select one from the partial listing of regions in *Table MW2–1*; we pick the North America entity. This re-engineering has the same pattern as the United States in the country example. Following this gives us the object schema in *Figure MW2–2*. Not surprisingly, this has the same shape as the United States schema in *Figure MW1-7*.

Figure MW2–3
Europe object schema



We now re-engineer another entity. We pick Europe from the partial listing of regions in *Table MW2–1*. We follow the same pattern that we used for United Kingdom in the country example and this gives us the object schema in *Figure MW2–3*— with the same shape as *Figure MW1-9*. Like the country example (see *Figure MW1-10*), we merge the two schemas and get *Figure* .

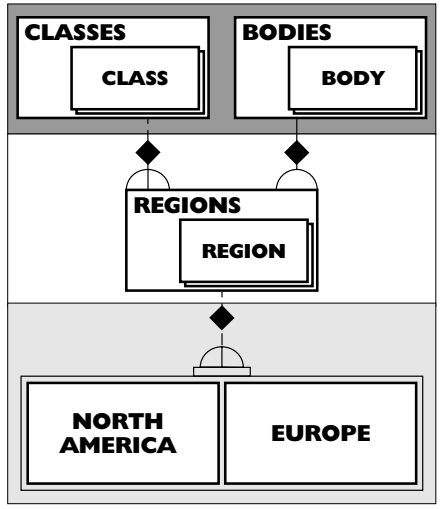
Figure MW2–4
Europe object schema



I Merged North America and Europe object schemat should be becoming clear how re-using patterns can simplify the re-engineering process. Even though we need to retrace each of the steps to confirm that the patterns are the same shape, it still makes the whole process much simpler.

2.3 Re-engineering the entity type sign

Figure MW2–5
Regions object schema



The re-engineering of the two individual regions reveals similar patterns; so, we use them (and the country entity type pattern) as a basis for re-engineering the region entity type. We get the object schema shown in *Figure MW2–5*; as you have by now come to expect, this is the same shape as the corresponding schema for the countries object in *Figure MW1-10*.

2.4 Re-engineering region full name attribute type sign

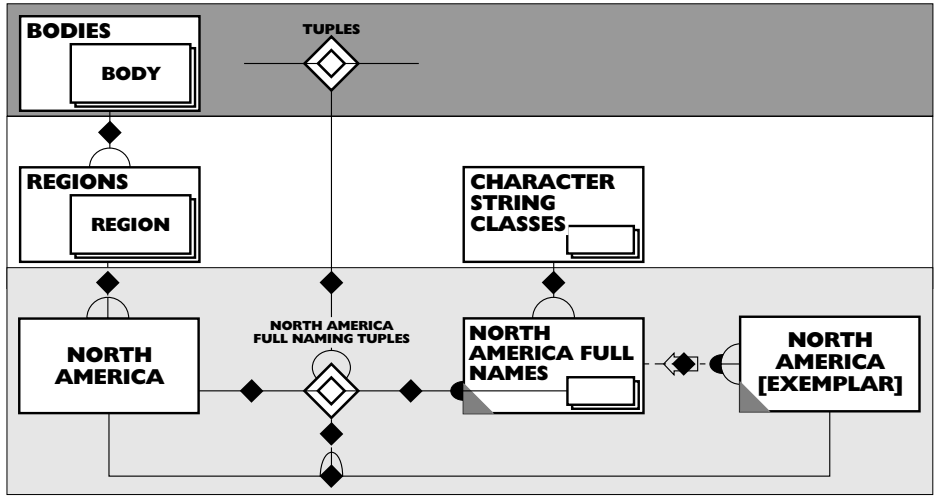
The re-engineering of the two region attribute type signs follows the same pattern as the country example. We follow the rules and start by re-engineering individual attributes and build up to the attribute type.

2.4.1 Re-engineering the attribute signs

We start with the entity North America's full name attribute, 'North America'. Following the country pattern, we get the object schema in *Figure MW2–6*. This is the same pattern as *Figure MW1-27*, a part from character string classes, which

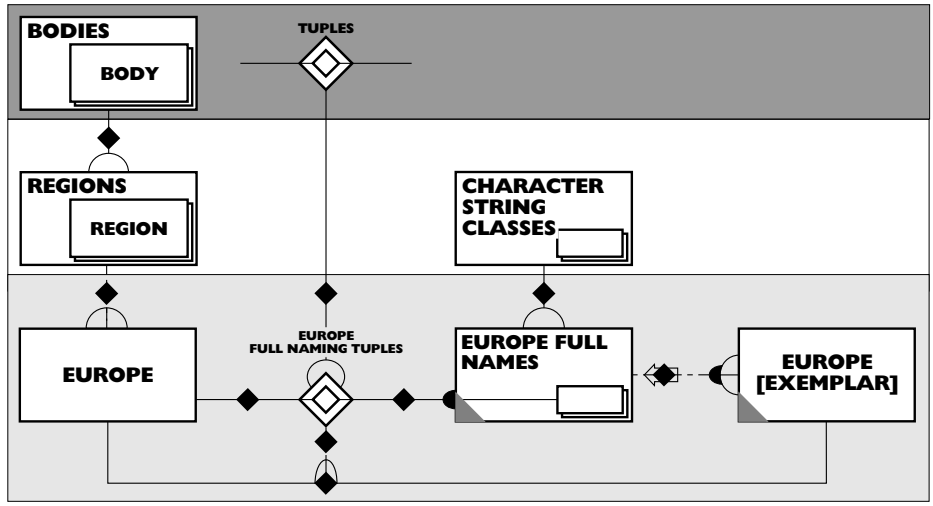
was re-engineered later. We included it in this schema to bind the objects more closely together.

Figure MW2–6 North America full naming tuples object schema



We check the patterns by re-engineering another attribute. We pick 'Europe—Europe's full name attribute. Again we follow the same country pattern, giving us the object schema in *Figure MW2–7*. As with *Figure MW2–6*, the only difference between this and its country equivalent is the inclusion of character string classes.

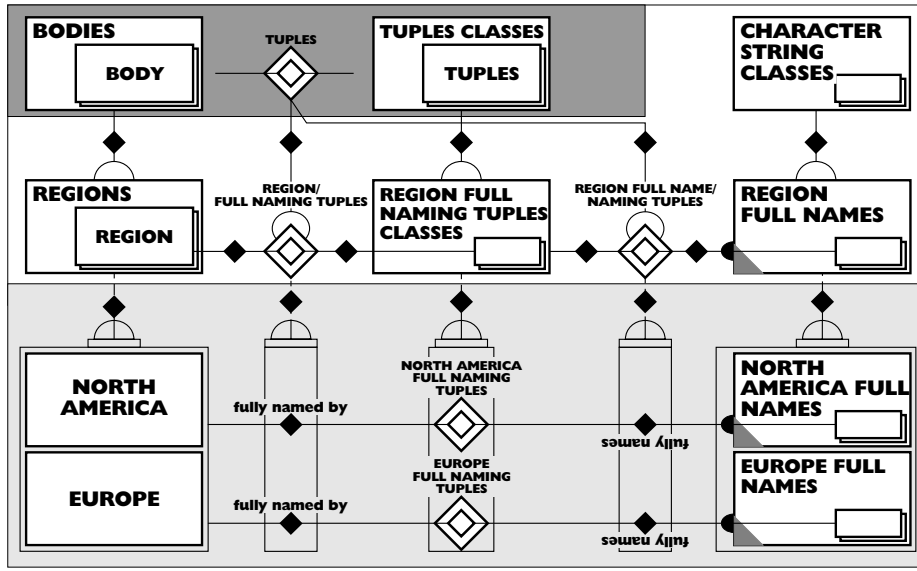
Figure MW2–7
Europe full naming tuples object schema



2.4.2 Re-engineering the attribute type sign

Following the rules and the country pattern, we now re-engineer the region full name attribute type sign into the region full names class and the region full naming tuples classes. We end up with the object schema in *Figure MW2–8*, which shows the same pattern as *Figure MW1-33*.

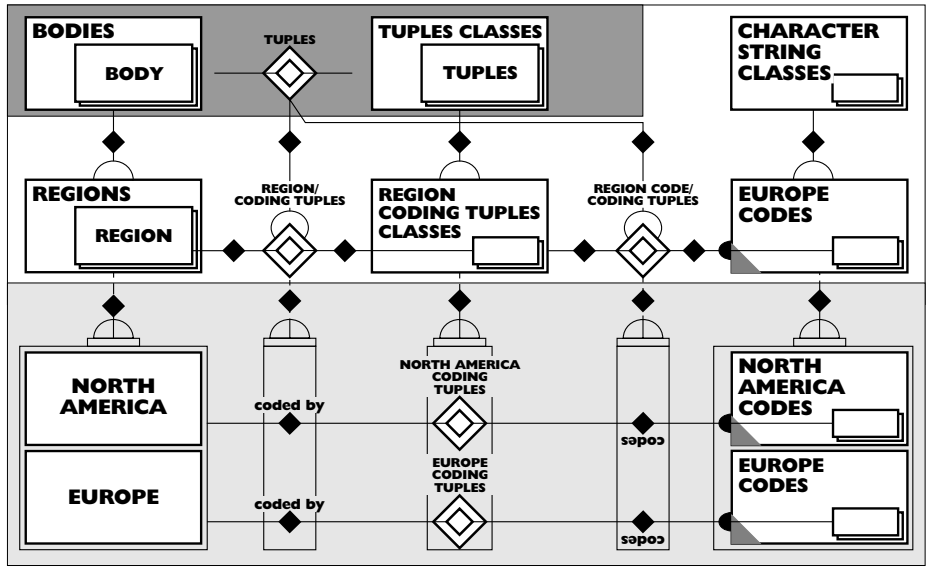
Figure MW2-8
Region full naming tuples classes object schema



2.5 Re-engineering region code attribute type sign

The region code attribute type follows both the country pattern and the region full name pattern (in *Figure MW2-8*). We go straight to the final object schema (shown in *Figure MW2-9*).

Figure MW2–9
Region coding tuples classes object schema



2.6 Generalising within region

Now that we have re-engineered the entity formats, we exploit the opportunities for generalisation. These fall into two groups:

- Within region, and
- Across region and country.

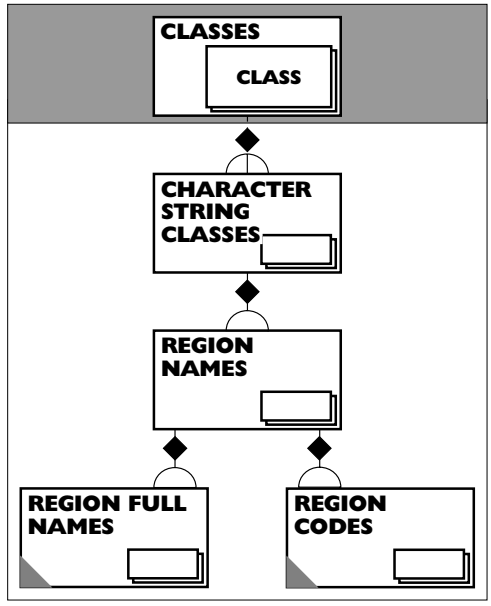
The first group shares the same generalisation patterns as the country example. The second group, which ranges across both the region and country sections of the model, as based on the generalisation of region and country into geo-political area.

We now look at the first group. There are generalisation patterns for:

- Region names,
- Exemplar region names, and
- Region naming tuples classes.

2.6.1 Generalising to region names

Figure MW2–10
Generalised
region names
object schema

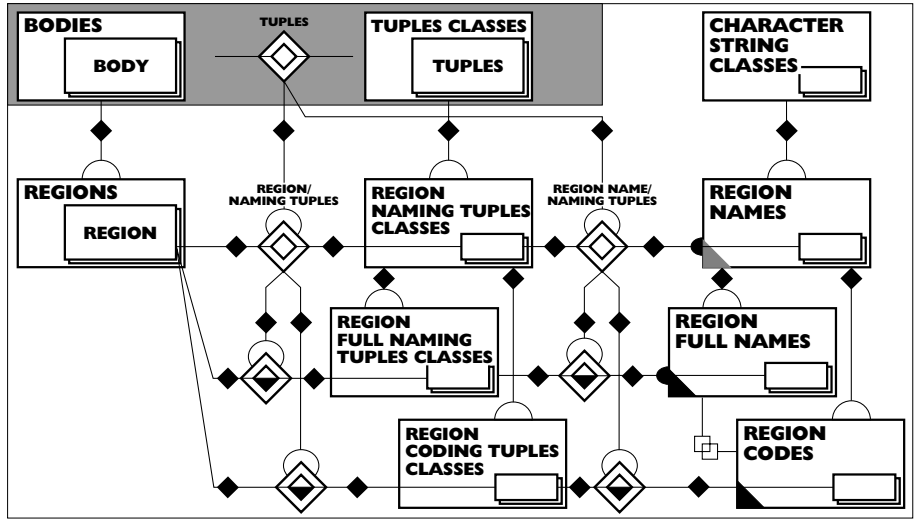


Following the country re-engineering pattern, we generalise region full names and region codes into region names (shown in the schema in *Figure MW2–10*.)

2.6.2 Generalising to region naming tuples classes

We follow the country pattern and generalise region full naming and coding tuples classes into region naming tuples classes (shown in the schema in *Figure MW2–11*). Now that we have the general region naming tuples classes, we have also generalised the region full naming and coding tuples classes class places up the super–sub-class hierarchy to region name/naming tuples and region/naming tuples. We have also classified region full names and codes as redundant. We can now, if we wish, purge them, compacting the model.

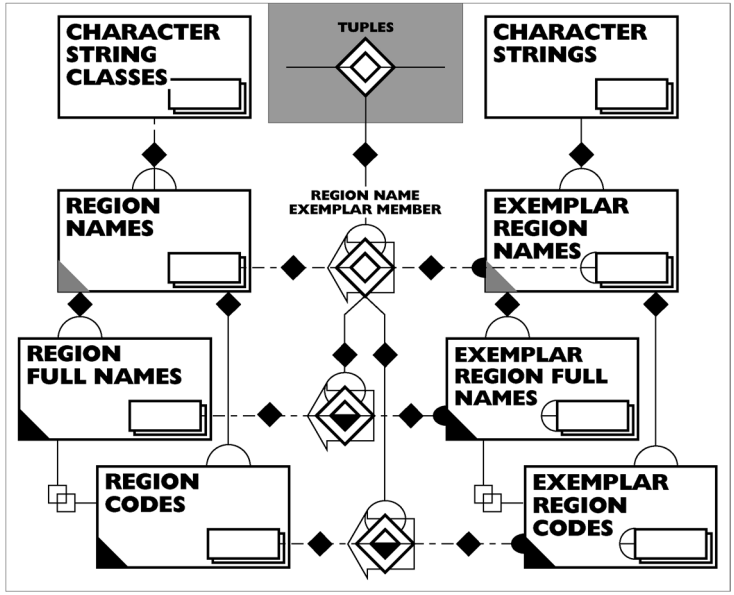
Figure MW2–11
Generalised region naming tuples classes object schema



2.6.3 Generalising to exemplar region names

We follow the pattern for exemplar country names (shown in MW1's *Figure MW1–41*) and generalise exemplar region full names and codes into exemplar region names. We also classify exemplar full names and codes as redundant. This gives us the schema in *Figure MW2–12*.

Figure MW2–12 Generalised exemplar region names object schema



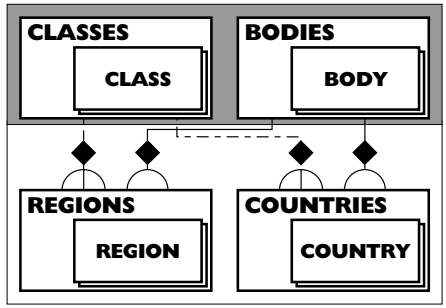
2.7 Generalising across region and country

Because we used the same patterns to re-engineer the entity formats of both regions and countries, an opportunity potentially exists to generalise across country and region.

2.7.1 Generalising to geo-political areas

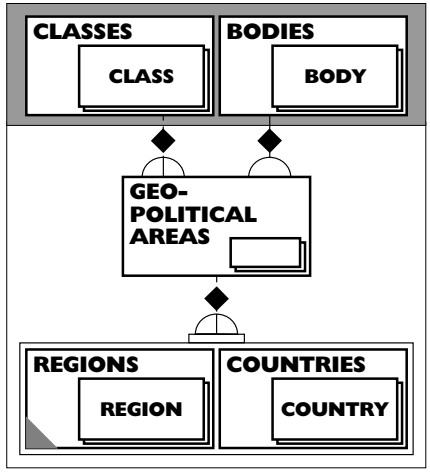
The opportunity is real. Country and region can be generalised to a super-class, geo-political area. To help us see the generalisation more clearly we look at the schema before and after generalisation (shown in *Figures MW2–13* and *MW2–14*).

Figure MW2–13 Before generalised regions and countries object schema



Before the generalisation, countries and regions are both separately connected to the framework level objects, classes and bodies. After the generalisation, geo-political areas is connected to these framework level objects, with regions and countries connected to geo-political areas.

Figure MW2–14 After generalised regions and countries object schema



There is one final bit of re-engineering with regard to what a region is. It is really no more than a geo-political area that is not a country. This means it is derived from the country class. We reflect this in the object schema in *Figure MW2–14*. Regions cannot be classified as redundant because it still has a number of connecting patterns depending on it, such as region naming tuples classes and region names. When we have generalised these, we will classify it as redundant.

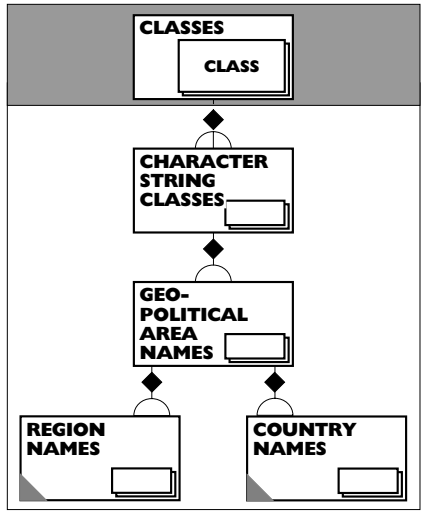
2.7.2 Generalising to geo-political area names

This generalisation of countries and regions to geo-political areas creates further opportunities for generalising their connecting patterns. Where countries and regions have similar connecting patterns, these can now be generalised into a single higher level connecting pattern for geo-political areas. We do this for the following connecting patterns:

- Geo-political area names,
- Geo-political area full naming and coding tuples classes,
- Exemplar geo-political area names, and
- Geo-political area naming tuples classes.

We start by generalising region and country names into geo-political area names. The result is shown in *Figure MW2-15*. This generalisation pattern is not new. It is exactly the same shape as the generalisation of country full names and codes to country names in MW1'S *Figure MW1-38* (and so the generalisation of region full names and codes to region names in *Figure MW2-10*). The region and country names are classified as derived. Later on, when we re-engineer their naming tuples classes, removing their dependent connections, we will classify them as redundant.

Figure MW2–15 Generalised geo-political area names object schema



2.7.3 Generalising geo-political area full naming and coding tuples classes

As the object model now stands, we still use separate full naming and coding tuples classes for country and region. It makes sense for us to generalise each of these up to a single class at the geo-political area level. This is the next step in moving the patterns for 'naming' from countries and regions up to the geo-political area level.

We generalise the region and country full naming tuples classes first. The result is shown in *Figure MW2–16*. Then we follow the same pattern and generalise the region and country coding tuples classes. The result is shown in *Figure MW2–17*. As you can see, the generalisation of the country and region level tuples classes to geo-political area level has enabled us to classify them as redundant.

Figure MW2–16
Generalised geo-political area full naming tuples classes object schema

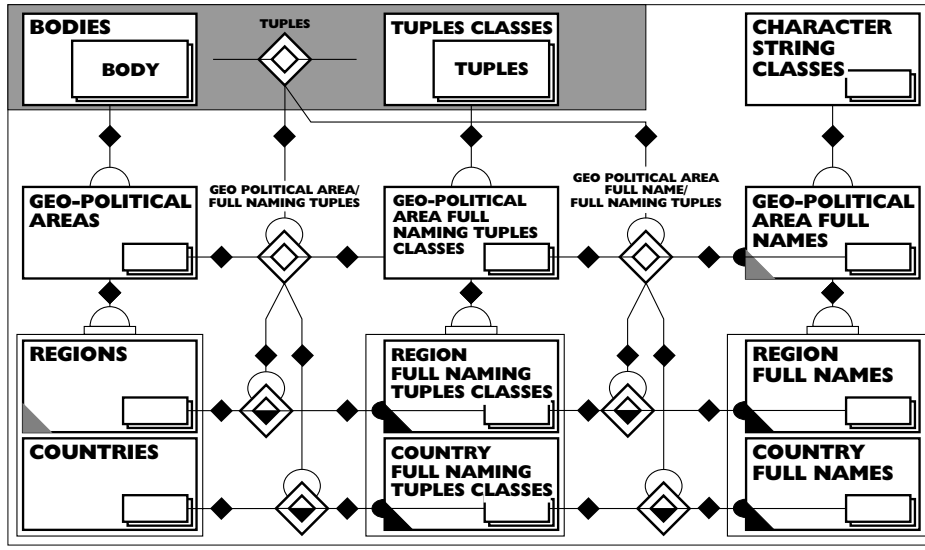
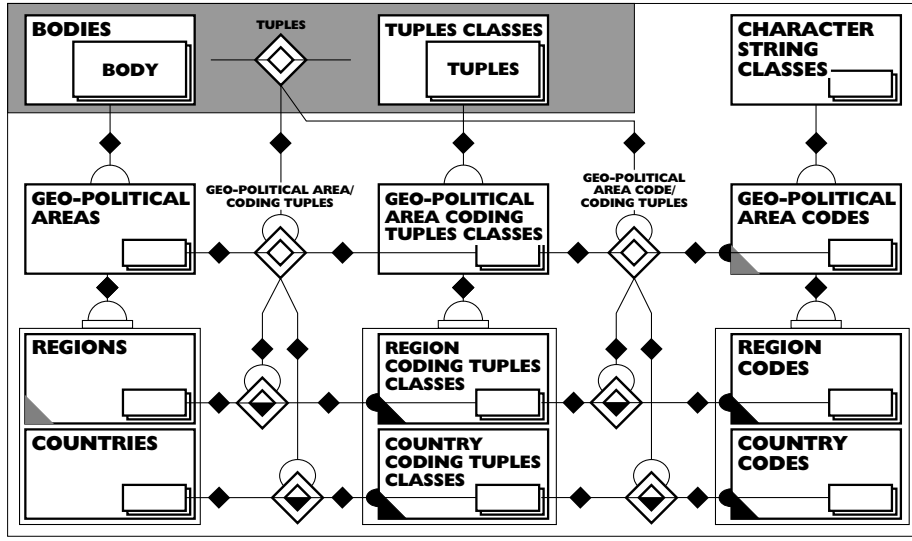


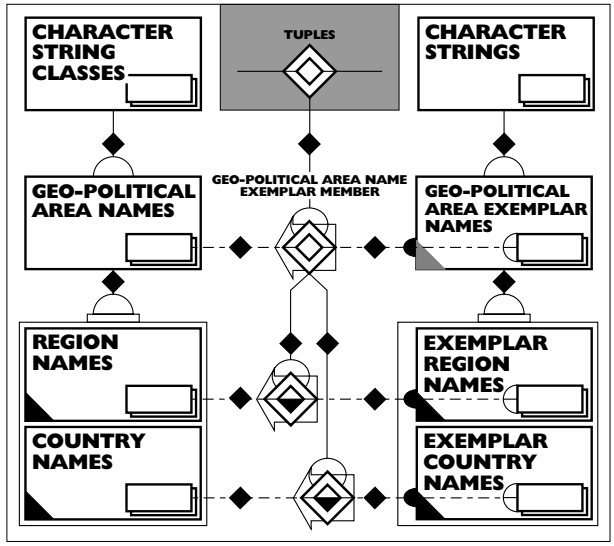
Figure MW2–17
Generalised geo-political area coding tuples classes object schema



2.7.4 Generalising to exemplar geo-political area names

We can generalise exemplar names to the geo-political area level, as shown in *Figure MW2–18*

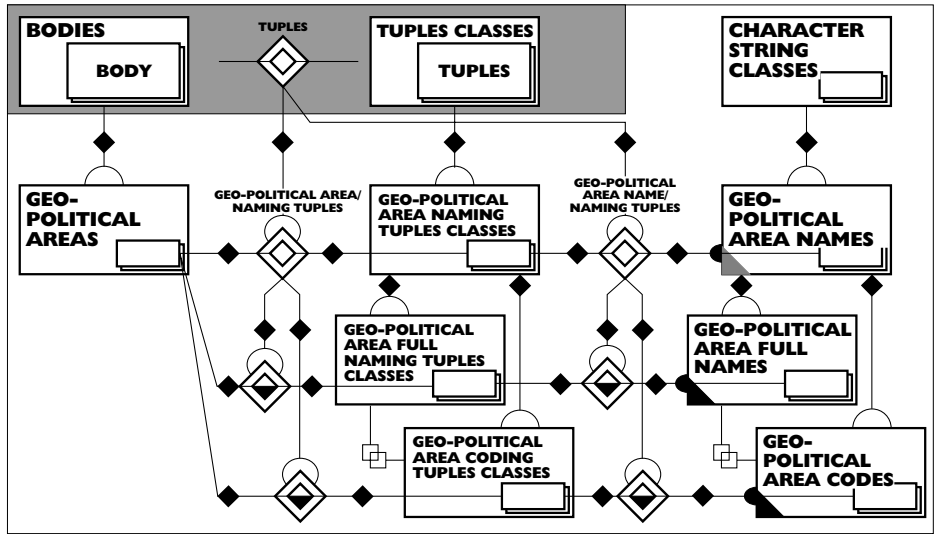
Figure MW2–18 Generalised exemplar geo-political names object schema



2.7.5 Generalising to geo-political area naming tuples classes

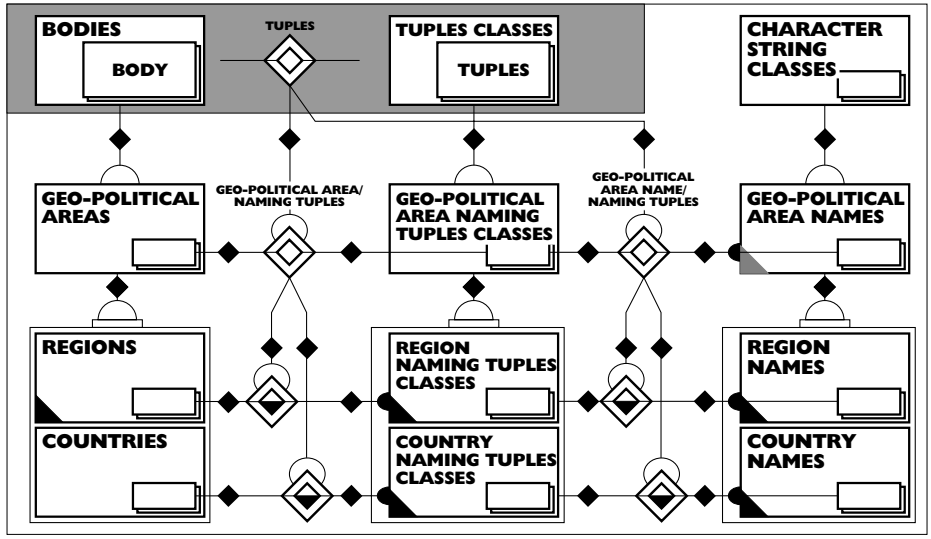
Finally we generalise to geo-political area, naming tuples classes from the geo-political area full naming tuples classes and geo-political area coding tuples classes (shown in *Figure MW2–19*). This has a similar pattern to the naming tuples classes generalisation schemas for country and region. Following the region pattern (shown in *Figure MW2–11*), we classify the codes and full names objects redundant.

Figure MW2–19 Generalised geo-political area naming tuples classes object schema—1



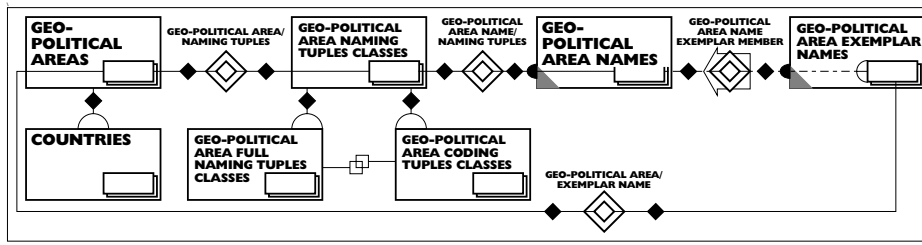
We can also follow a different generalisation route to the same geo-political area naming tuples classes, by generalising the country and region naming tuples classes. This gives us the object schema in *Figure MW2–20*. With the generalisation to geo-political area names, we classify the lower level region and country names redundant. Our classification of the region naming tuples classes object as redundant means that the regions object no longer has any dependent connections; so, we have also classified it as redundant.

FigureMW2-20 Generalised geo-political area naming tuples classes object schema—2



The generalisation has significantly compacted the model. The application level now has only eleven non-redundant objects. These can all be fitted onto one schema—*Figure MW2-21*. You may notice that the character strings patterns are not included in this schema; this is because they are not part of the first stage re-engineering. They were initially constructed as part of the second stage of the country re-engineering.

FigureMW2-21 First stage application level object schema



2.8 Re-engineering the region entity formats

This completes the first stage—re-engineering the existing system's region entity formats. The generalisation of the names and region patterns to the geo-political level is a good illustration of how patterns from previous re-engineerings

are re-used to both simplify the re-engineering process and provide opportunities for generalisation.

The example has generated high levels of generalisation, re-use and compacting. What is also interesting is seeing the large number of patterns that have been re-used from the re-engineering of country.

It also shows quite clearly that the re-engineering of the entity formats is not just a translation of the existing entity patterns into an object semantics. For instance, you will not find the patterns for the comprehensive view of countries and regions as geo-political areas, and the general view of names across all three, in the existing entity system. The systematic re-engineering process takes patterns from an existing entity system and produces a more general object model.

3 Re-engineering our conceptual patterns for region

We now move onto the second stage of the re-engineering—capturing our conceptual patterns for region. The previous country worked example (MW1) provided us with an illustration of the process for investigating our conceptual patterns. It showed us the steps that we need to go through to identify interesting ones and the process of re-engineering them into the object model.

If this was a 'real' re-engineering, and not an exercise, we would go through a full investigation of all region's conceptual patterns. Instead, we are just going to look at how the nesting and stages patterns re-engineered in the country example can be re-used on region.

We apply country's two patterns to region. Then (as with the patterns in the first stage of region's re-engineering), we generalise them and their corresponding country patterns up to the geo-political areas level. It turns out that all these patterns can be generalised into a single geo-political areas pattern.

3.1 Nested regions

When we re-engineered country's nesting pattern in *MW1—Re-Engineering Country* (illustrated in its *Figure MW1-53*, I noted that it looked as though it could be re-used for other types of areas. It can be; we now look at two examples:

- Region nested within region, and
- Country nested in region.

We start off, as we did in the country example, by assuming that the individual nesting patterns are between individual regions and countries—not stages. We do not have region's stages pattern until we re-engineer it in the next section.

3.1.1 Region nested in region

We start with the region nested in region pattern. The original sample of regions we listed in *Table MW2-1* are mutually exclusive; they do not nest. In fact, most computer system's regions pattern assumes that regions do not nest. We came across a similar assumption when we re-engineered country's nesting pattern. Both assumptions are made for the same reason; the systems use both countries and regions as hooks to summarise figures. Typically, countries provide the first level of summarisation, regions the second.

However in 'reality' we can and do have nested regions, just as we can and do have nested countries. For the re-engineering, we take a revised sample of regions, one

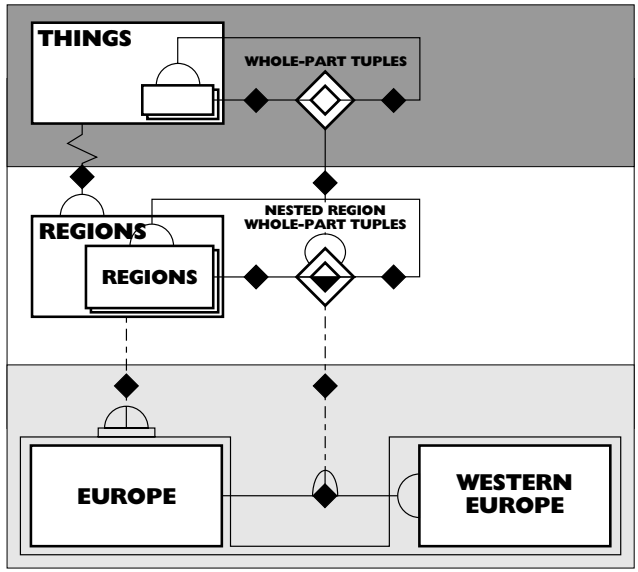
that provides us with examples of nesting. This is given in *Table MW2–3*. The new regions are shaded.

Table MW2–3: Revised partial region listing

Region Name	Region Code	Nesting in Region
Europe	EU	
European Community	EC	
Far East	FE	
Middle East	ME	
North America	NA	
South America	SA	
Western Europe	WE	Europe

When we re-engineer the Western Europe nesting in Europe pattern in *Table MW2–3*, we get the object schema in *Figure MW2–22*. This has the same pattern as nested country whole–part tuples (shown in MW1's *Figure MW1–53*). As in the's nested country pattern, the nested region whole–part tuples is redundant, derived from the regions class.

FigureMW2–22
Region nesting in region



3.1.2 Country nested in region

We now turn our attention to the nesting pattern from country to region. This pattern is likely to be an existing system pattern. Many computer systems, certainly in my experience, connect countries to regions. Normally their country entity format has a 'belonging to region' attribute type that relates each country to one region—like the one shown in *Table MW2–4*.

Table MW2–4: Revised country entity format

Entity Type	Attribute Type #1	Attribute Type #2	Attribute Type #3	Etc.
Country	Country name	Country code	Belonging to Region	-

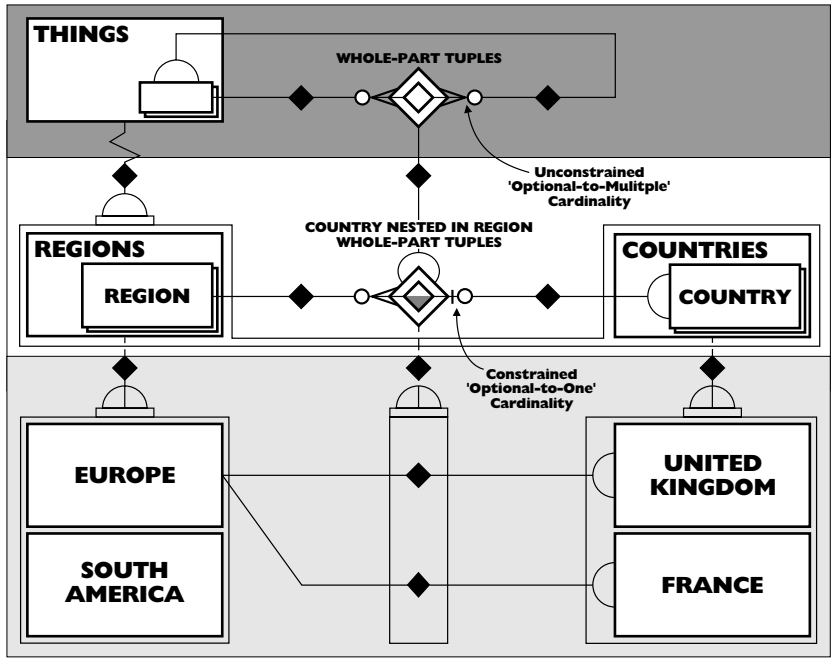
If we update the original Partial Country Listing (MW1's *Table MW1–1*) with the 'new' attribute type, we get the extended partial country listing in *Table MW2–5*.

Table MW2–5: Extended partial country listing

Country Names	Country Codes	Belonging to Regions
Germany	DM	EU (Europe)
Italy	IT	EU (Europe)
Japan	JP	FE (Far East)
Turkey	TK	ME (Middle East)
United Kingdom	UK	EU (Europe)
United States	US	NA (North America)

If we re-engineer this, we get the, by now familiar, nesting tuples pattern, this time between the countries and regions classes (shown in the schema in *Figure MW2–23*). You may have noticed that the country nested in region whole–part tuples class is shown as derived—rather than redundant as in the earlier versions of the nesting pattern (for example, *Figure MW2–22*). This is because the tuple class has additional information in its cardinality faithfully reflecting the constraints in the entity format in *Table MW2–4*. The cardinality linking it to countries is constrained to optional-to-one; in other words, each country can only 'belong to' one region. This is because (as is discussed in some detail in *OP1—Entity Ontology Paradigm*) attributes, such as 'belonging to region' cannot reflect 'many-to-many' connections.

FigureMW2-23
Country nested in region object schema



If you look carefully at the country listing in *Table MW2-5* and the region listing in *Table MW2-3*, you should be able to spot how the entity format constraint has made the listing incomplete. Something is missing for Germany, Italy and the United Kingdom. They are not only part of Europe but also part of the European Community.

One simple workaround that may spring to mind is to make the European Community (EC) part of Europe (this gives a tree structure). But the example has been constructed so that this does not work. Turkey, a potential member of the EC (it has applied), is not part of Europe. In other words, Europe does not necessarily contain all the European Community (it will not when Turkey joins the EC). And vice versa, the European Community does not yet contain all of Europe.

So an accurate model needs to be able to reflect European EC members, such as Germany, as nested in at least two regions: Europe and the EC (this gives a lat-

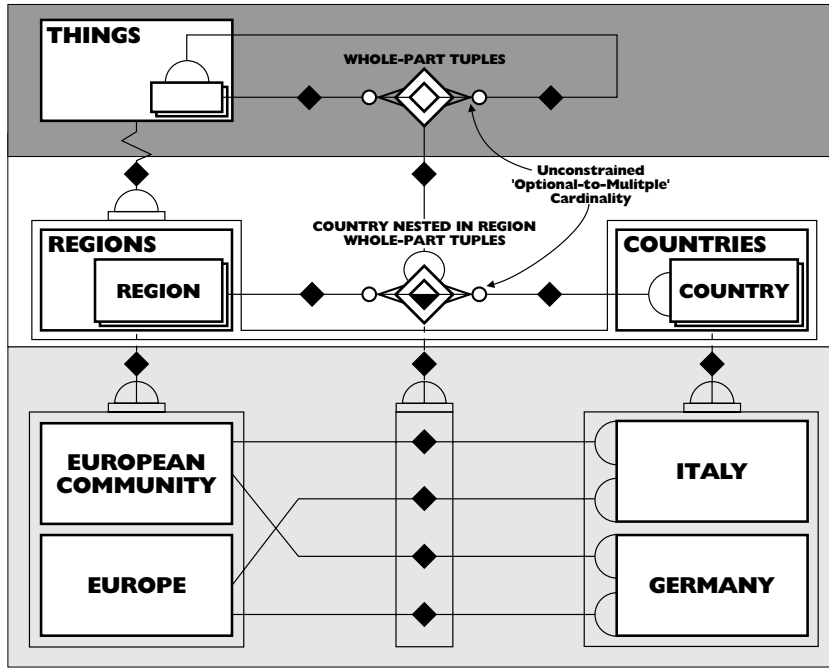
tice structure). We revise the extended partial country listing in *Table MW2–5* to reflect this. The result is shown in *Table MW2–6*.

Table MW2–6: Revised extended partial country listing

Country names	Country Codes	Belonging to Regions
Germany	DE	EU (Europe)
		WE (Western Europe)
		EC (European Community)
Italy	IT	EU (Europe)
		WE (Western Europe)
		EC (European Community)
Turkey	TK	ME (Middle East)
		EC (European Community)
Japan	JP	FE (Far East)
United Kingdom	GB	EU (Europe)
		EC (European Community)
United States	US	NA (North America)

We reflect this insight in the object model by relaxing the cardinality from optional-to-one to optional-to-multiple (shown in *Figure MW2–24*). If you think about it, you will realise that this less constrained cardinality applies to the other nesting patterns (country nested in country and region nested in region) as well. You may have noticed that the relaxation of the cardinality has another effect. The country nested in region whole–part tuples class is now classified redundant—like the other nesting tuples classes. This is because its cardinalities no longer contain any additional information.

FigureMW2-24 Unconstrained country nesting in region object schema



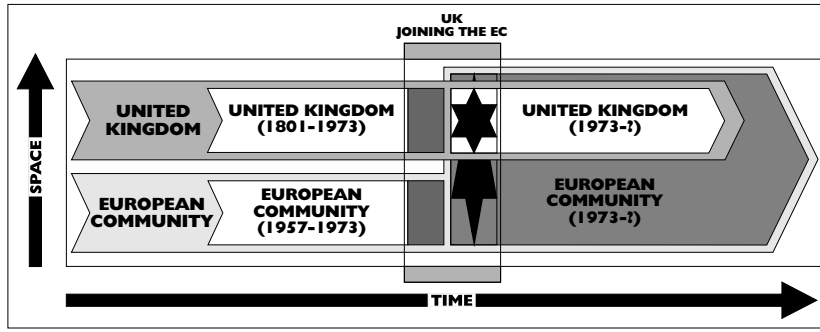
3.2 Region stages

We now extend the nesting pattern to include region stages, following the country pattern. We start by re-engineering one of the events that cause these stages, country joining region. This gives us the region stages pattern, which we use to generalise the nested pattern to the geo-political areas level.

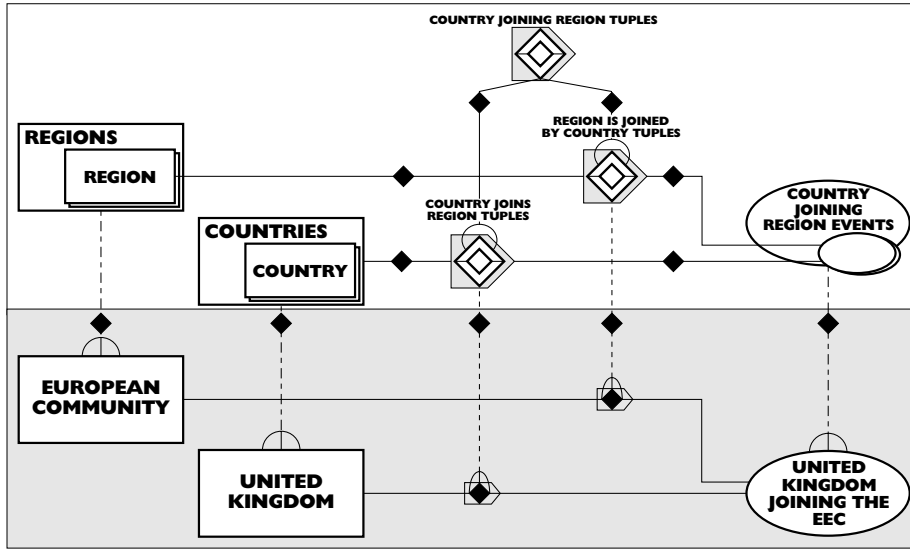
3.2.1 Re-engineering the country joining region event pattern

We start with an example; the United Kingdom joining the European Community in 1973. The re-engineering gives us the space-time map shown in *Figure MW2-25* and the object schema shown in *Figure MW2-26*. These have similar shapes to the country re-engineering example; the 1707 Act of Union event joining Scotland to the United Kingdom (shown in MW1's *Figure MW1-54* and *Figure MW1-55*). This bodes well for the generalisation.

FigureMW2-25 United Kingdom joining the European Community space-time map

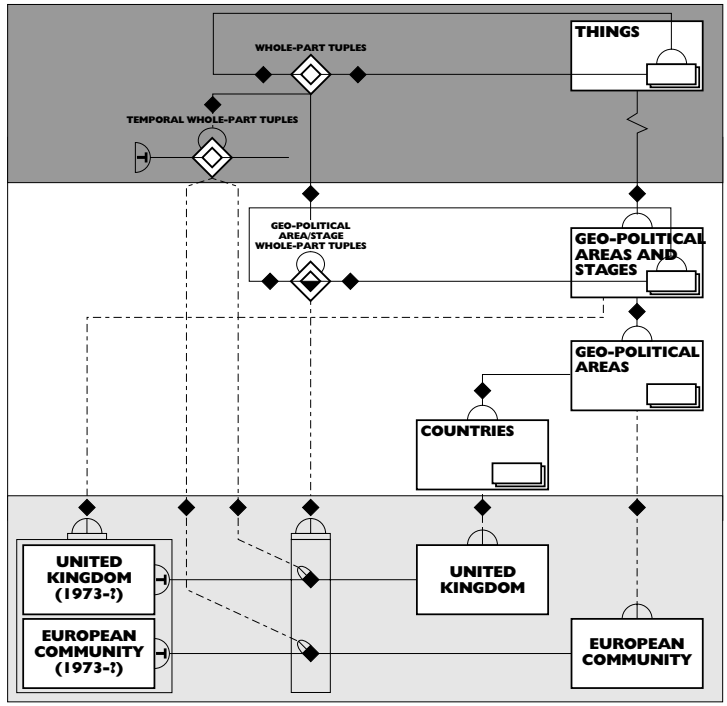


FigureMW2-26 Country joining region events object schema



We then take the pattern up to class level. First for the country joining region events, giving us the schema in *Figure MW2-26* And then for the country and region stages, giving us the schema in *Figure MW2-27*. Note that this combines the nesting and stages patterns and generalises them to the geo-political areas level.

FigureMW2–27
Generalised
geo-political
areas & stages



3.2.2 Generalising to geo-political areas joining event

Our concern here is not with joining events. We only use them to provide us with an example of the region stages pattern. However, it should be clear that the country joining region event pattern is one of the geo-political area joining event patterns. And that country's Acts of Union pattern (see MW1's *Figure MW1–59*) is also one, although they have slightly different shapes. Unlike the country joining region event pattern, the joining in the Acts of Union pattern constructs a new country. In a 'real' re-engineering, we would re-engineer all the different patterns and then generalise them to the geo-political area joining event, raising the pattern up to the geo-political areas level. But we have done all we need to do for this worked example.

3.2.3 Geo-political area nesting/stage pattern re-use

If we consider the re-engineering so far, three main patterns have emerged. These are:

- The naming pattern,
- The geo-political area joining pattern, and
- The combined geo-political area nesting/stage pattern.

All of these provide us with good examples of the power of re-engineering. Take the last pattern—geo-political area nesting/stage. The application level (the one that matters for system building) of the object model in *Figure MW2-27* is both more general and functionally richer than the system from which it was re-engineered. It operates at the geo-political area level, and unlike the entity format, it can reflect:

- Countries nested in countries,
- Regions nested in regions, and
- Countries nested in multiple regions.

In the next worked example, this pattern will be re-used again, generating more compacting.

4 Summary

Our object model for spatial patterns is now more general and more powerful after the generalising of country's conceptual patterns across regions and up to geo-political areas.

This example has shown, again, how much potential there is for re-use and generalisation. The way in which country and region share patterns, and the ease with which these patterns combine, is uncanny. However, this can be explained in terms of our strong unconscious conceptual patterns for space. Uncovering these and

making them explicit and accurate in an object model is the purpose of the re-engineering.

The re-engineering of spatial patterns is not yet complete. In the next worked example (*MW3— Re-Engineering Bank Address*), we re-engineer the final example of a spatial pattern—address. However, this will demonstrate the re-usability of our general, object reference ontology, rather than enhance it.

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

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

MO1—*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?*

RE-ENGINEERING REGION

A-S

A

application level (of model) ------ MW2-19, MW2-29
attribute
re-engineer
order ---------- MW2-2

C

compacting
purging redundant objects ---------- MW2-10

E

entity
re-engineering order rules ---------- MW2-2
explicit
business model ---------- MW2-30

F

framework level (of model) ---------- MW2-13

R

redundant patterns ---------- MW2-22
classifying --- MW2-10–MW2-13, MW2-17–MW2-18, MW2-25
re-use
country/region patterns ----- MW2-2, MW2-4, MW2-29

S

structure – lattice and tree ---------- MW2-24
tree structure, *See structure – lattice and tree*
super–sub-class hierarchy ---------- MW2-10