B usiness
O bject
R eference
O ntology
Program

simplifying semantics

Working Paper

MW4

Methodology: Worked Example 4

Re-Engineering Time

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

MW4

Methodology: Worked Example 4

Re-Engineering Time

CONTENTS

CONTENTS
MW4

BORO

MW4

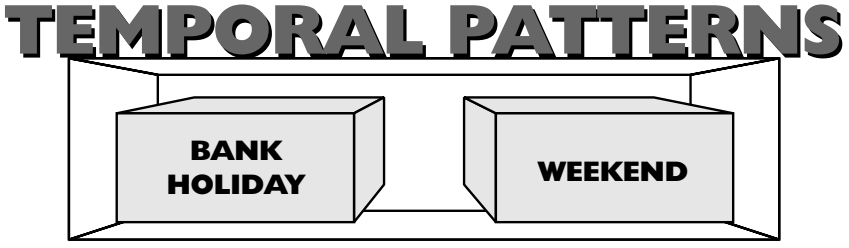
Methodology: Worked Example 4

Re-Engineering Time

1 Introduction

In the last three worked example papers, we constructed a reference object ontology model for spatial patterns by re-engineering low-level three entity formats. We now turn to the similar task of constructing a generalised reference objectontology model for temporal patterns. We do this by re-engineering the two low-level entity formats; bank holiday and weekend. Their re-engineering reveals temporal patterns that are very general and will be found in most systems. Now, that we are familiar with the steps in the re-engineering process, we focus on what is being re-engineered.

Figure MW4–1
Two examples of temporal patterns



The spatial patterns that we re-engineered in the previous examples were reasonably intuitive. The analysis can be seen as, in some ways, a clarification of our

common-sense views of space. The re-engineering of time is different. The object paradigm offers such a radically different view of time and temporal patterns that our intuitions cannot give us any sensible guidance (as we saw when we examined its semantics in *OP4—Business Object Ontology Paradigm*). Even though the temporal model we re-engineer in this paper provides a good explanation of our everyday uses of temporal terms, the patterns themselves initially seem odd. This is a good sign; it is to be expected from a radical re-engineering.

2 Re-engineering an existing system's bank holiday entity format

We start by re-engineering the bank holiday entity format. As usual we start by looking at some of the existing entities. *Table MW4–1* gives us a Partial Bank Holiday Listing.

Table MW4–1: Partial bank holiday listing

Country Code	Date
UK	09-Apr-93
UK	03-May-93
US	18-Jan-93
US	16-Feb-93
US	31-May-93

From this, we can deduce the entity format shown in Table MW4–2.

Table MW4–2: Bank holidays entity format

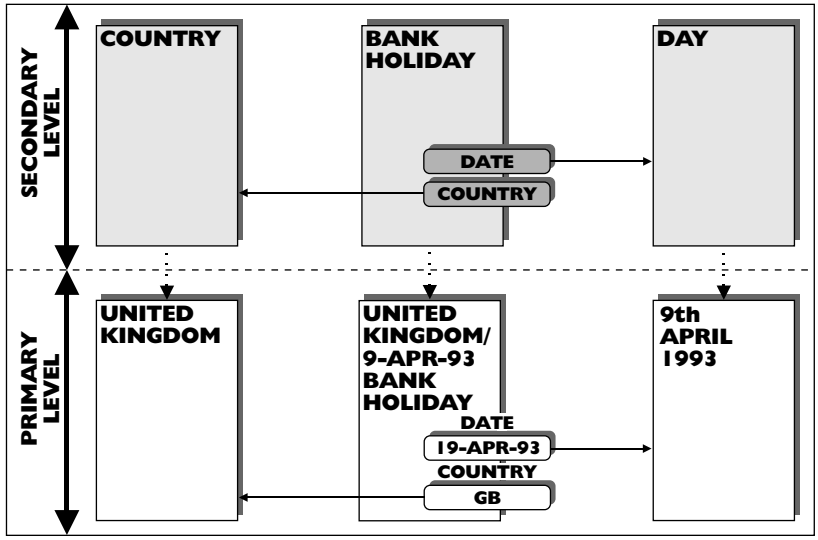
Entity Type	Attribute Type #1	Attribute Type #2
Bank holidays	Country code	Date

2.1 Bank holiday as a relational entity

Bank holiday is a relational entity type. It is derived from the entity types, country and day. It is not a pseudo entity type, such as the employee–project we discussed in *OP1—Entity Ontology Paradigm* (see its *Figure OP1–25*). Those were a

workaround solution to the problem of handling many-to-many relational attribute types in the entity paradigm. Unlike them, bank holiday is real. We will confirm this when we see the object that it is transformed into. But, like the pseudo entity types, bank holiday's two attribute types link it to the entity types from which it is derived (illustrated in).

Figure MW4–2
Bank holiday relational entity



When re-engineering a real or pseudo relational entity type, we follow a similar rule to re-engineering a relational attribute. We re-engineer the related entity types before we re-engineer the relational entity type. So here we re-engineer country and day before we re-engineer bank holiday. Country has already been re-engineered in a previous example; so, we only need to re-engineer day.

3 Re-engineering day

Most computer systems do not have a day file (corresponding to the entity type shown in) with a record (entity) for each day. In other words, the day (or date) entity is not usually found explicitly implemented in most computer systems. However, it is there, but implemented implicitly. The individual day entities are

deduced from the day code attributes of other entities. It can be thought of as an entity implemented as a process.

We need to re-engineer this implicit entity. The easiest way to do this is to construct an explicit entity format for it and subject this to the standard re-engineering procedure. Once the format is constructed, we can continue as usual. We familiarise ourselves with the entity format by looking at some examples of what we are going to re-engineer. Table *Table MW4–3* gives us a partial day listing.

Table MW4–3: Partial day listing

Day Code
09-Apr-93
10-Apr-93
11-Apr-93
12-Apr-93
13-Apr-93
14-Apr-93

From this, we deduce the entity format shown in *Table MW4–4*.

Table MW4–4: Day entity format

Entity Type	Attribute Type #1
Day	Day Code

3.1 Re-engineering the day entity type sign

In this example, we only re-engineer the day entity type sign. We do not need to re-engineer its associated day code attribute type sign. If we were to re-engineer it, then it would fall under the naming pattern re-engineered in the previous papers. For the day entity type sign, we follow the second rule and re-engineer a couple of day entities before the day entity type.

We start with an individual day selected from *Table MW4–3*—9th April 1993. From an entity point of view, the sign refers to a period of time—a day. It starts just after midnight of the 8th April 1993 and continues until midnight 9th April 1993.

The examination of the object semantics of temporal patterns, in *OP4—Business Object Ontology Paradigm*, tells us what object this entity is transformed into. It is a temporal stage of the whole of space-time. It is every part of space-time for the period that starts just after midnight of the 8th April 1993 and continues until midnight 9th April 1993. Because it persists through time, it is a physical body. A space-time map of the object is given in *Figure MW4–3*, an object schema in *Figure MW4–4*. Other days can be re-engineered into similar patterns.

Figure MW4–3 Space-time map for 9th April 1993

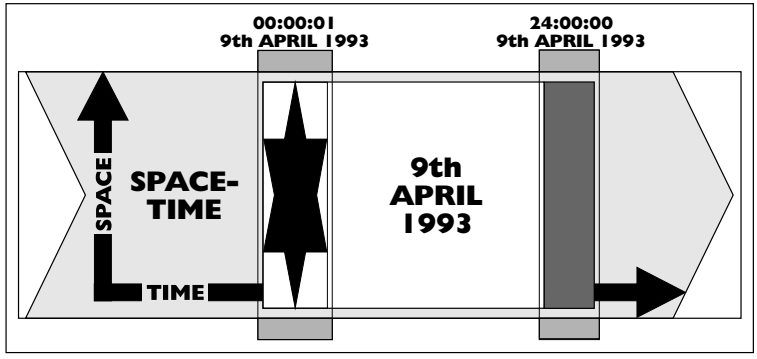
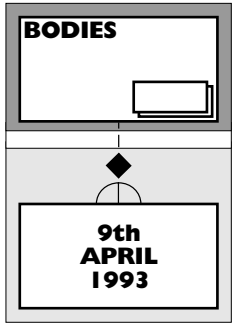


Figure MW4–4 9th April 1993 object schema



This is a very different way of seeing time, but we can translate the way we talk into it. We normally say 'today is the 9th April'. This can be understood as 'now is a temporal part of the four-dimensional object 9th April'. This is a mouthful and so not a practical alternative for everyday conversation.

3.1.1 Re-engineering the entity type sign

We follow the normal re-engineering pattern. The entity type sign is re-engineered into the class of objects that the entity signs were re-engineered into. This gives us the class days. The members of this days class divide the four-dimensional space-time 'sausage' into regular day long slices. This is illustrated in the space-time map in *Figure MW4–5* and modelled in *Figure MW4–6*.

Figure MW4–5
The days class

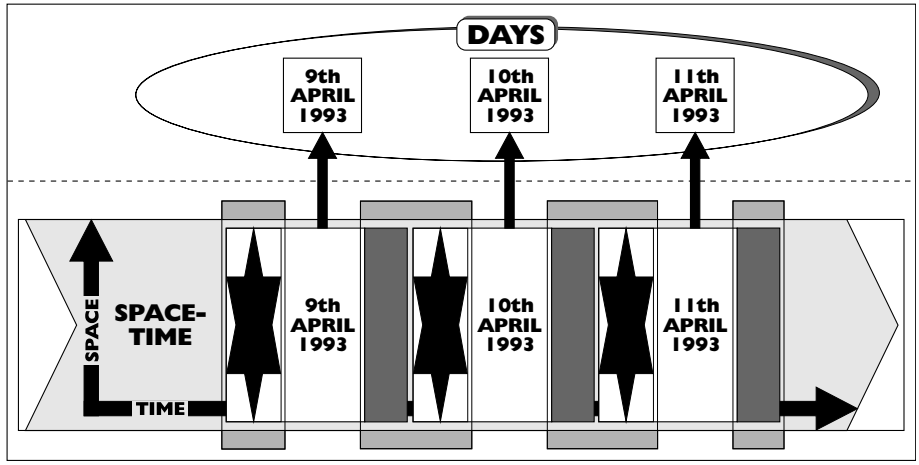
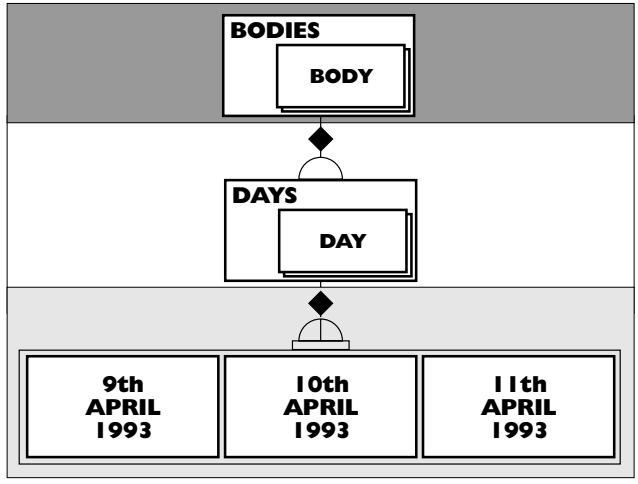


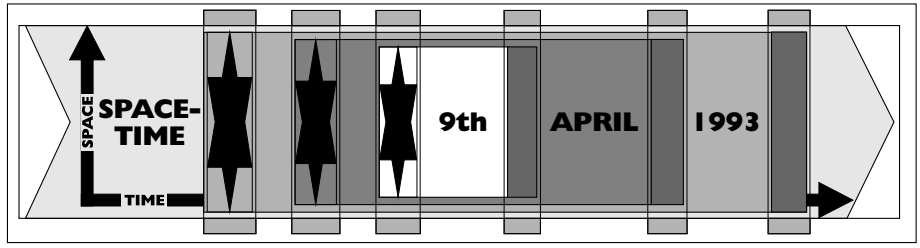
Figure MW4–6
Days object schema



3.2 Re-engineering other time periods

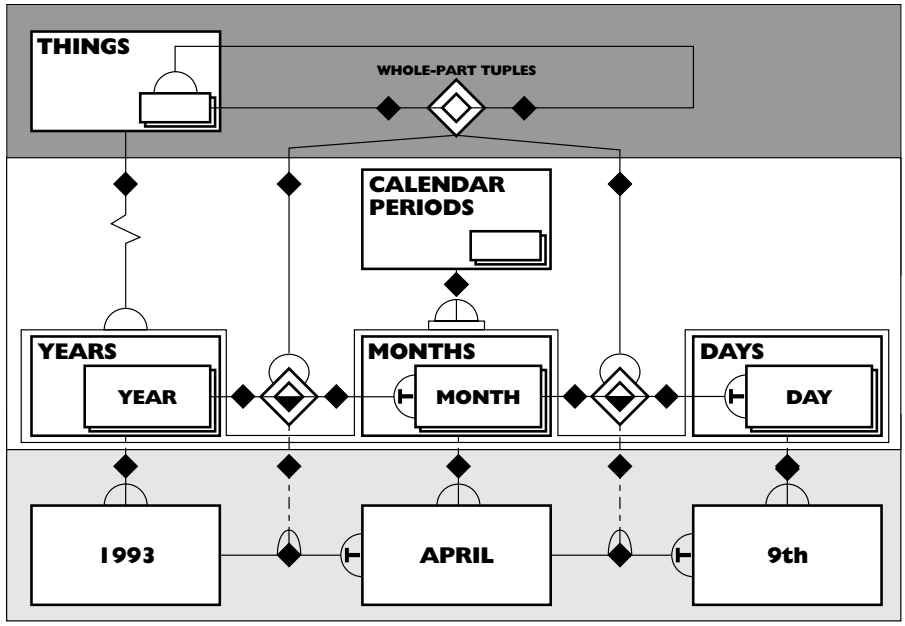
We can extrapolate this pattern to give an object-oriented view of other time periods, such as months and years. These divide the space-time 'sausage' into larger month and year long temporal slices (shown in the space-time map in *Figure MW4–7*). The smaller slices are temporal parts of the larger slices. So, in the object paradigm, a day is part of a month and a month part of a year in the same way as my hand is a part of my arm.

Figure MW4–7 Year, month and day space-time Map



All three classes can be regarded as members of a calendar periods class (shown in the object schema in *Figure MW4–8*). This gives us one of the groups of basic temporal patterns we need for our temporal object model. The remainder of the re-engineering will give us the rest.

Figure MW4–8 Calendar periods object schema



4 Re-engineering bank holiday

Now that we have re-engineered country and day, we can re-engineer the derived relational entity, bank holidays.

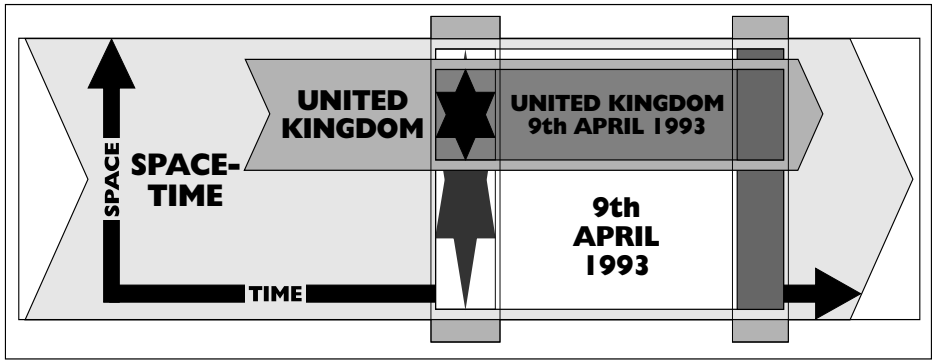
4.1 Re-engineering the bank holiday entity type sign

We follow our rules and start with an entity, an individual bank holiday. Let's start with the 9th April 1993 in the United Kingdom, the first entry in *Table MW4–1*. What does this refer to? From an entity point of view, it is a period of time—a day— that is a bank holiday in the United Kingdom.

We know what this bank holiday's related objects, United Kingdom and 9th April 1993, are. These give us a clue to what the bank holiday object is. It is the intersection (overlap) of the two objects the United Kingdom and the 9th April—in other words, that bit of space-time that is part of both objects. This is illus-

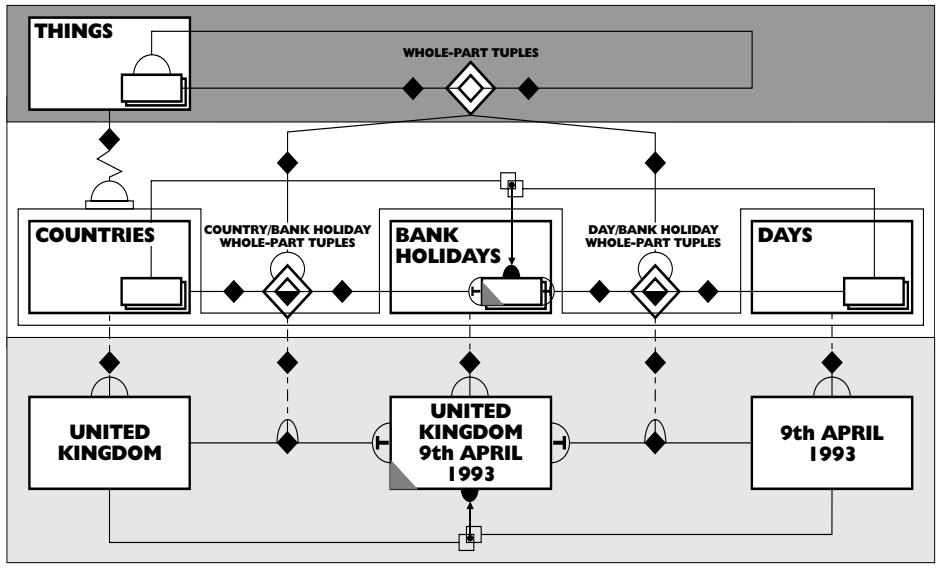
trated in the space-time map in *Figure MW4–9*. This shows that the intersected bank holiday object is constructed from the other two objects.

Figure MW4–9 United Kingdom's 9th April bank holiday space-time Map



We follow the standard pattern of re-engineering for the entity type sign. It re-engineers into the class of objects that its entity signs were re-engineered into; this is the bank holidays class. All the members of this class are logically dependent on the whole–part tuples that they have with members of the countries and days classes. This makes the members derived, but not the class itself. This is shown in the object schema in *Figure MW4–10*.

Figure MW4–10 Bank holidays object schema



Notice that the bank holiday's connections to the intersecting objects are whole–part; they are structural connections between extensions. These define the individual bank holiday object and so we classify it as derived. The re-engineering of the other bank holidays follows the same pattern.

It may seem odd that a bank holiday is a physical object. But it gives a good explanation of the way we talk. If we are travelling through a country on a bank holiday, we say, 'it is a bank holiday here'. In object-oriented English, this translates into 'my here object is part of the bank holiday object'. If we were to travel straight onto the next country, we would say, 'it is not a bank holiday here'. This translates as 'my here object is not part of a bank holiday object'.

4.2 State of the object model for temporal patterns

This completes the re-engineering of the bank holiday entity formats. It has given us a reasonable foundation for the object model of temporal patterns. The calendar periods are useful general patterns that can be re-used in most re-engineerings. The bank holiday pattern is lower level, but still reasonably useful. It also links us in to the spatial patterns' object model by way of countries. Later on in this paper, we will see how we can generalise this link up to geo-political areas.

5 Re-engineering an existing system's weekend entity formats

We now turn to the re-engineering of the second example—the weekend entity format. We start, as usual, by looking at the existing entities; a partial list is given in *Table MW4–5*.

Table MW4–5: Partial weekend listing

Country	Indicators						
Code	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
UK	No	No	No	No	No	Yes	Yes
US	No	No	No	No	No	Yes	Yes

From this, we get the entity format in *Table MW4–6*.

Table MW4–6: Weekend entity format

Entity Type	Weekend
Attribute Type #1	Country code
Attribute Type #2	Monday indicator
Attribute Type #3	Tuesday indicator
Attribute Type #4	Wednesday indicator
Attribute Type #5	Thursday indicator
Attribute Type #6	Friday indicator
Attribute Type #7	Saturday indicator
Attribute Type #8	Sunday indicator

5.1 Re-engineering the weekend entity type sign

We follow the rules and pick an individual weekend entity sign to re-engineer—the United Kingdom (coded as UK) from the partial listing in *Table MW4–5*. This indicates that the United Kingdom's weekend is Saturday and Sunday. Even though Saturday and Sunday appear as individual entities in this system, they are entity types. Their entities are individual Saturdays and Sundays—such as the 6th and 7th May 1995. This means that the United Kingdom entry we selected from *Table MW4–5* is also an entity type with entities.

A partial listing of these entities—particular United Kingdom weekends—is given in *Table MW4–7*. We start re-engineering at this level and work our way up to the selected entity sign. We select an example weekend—the 6th/7th May 1995—to re-engineer.

Table MW4–7: Particular United Kingdom weekends partial listing

Weekends	Dates	Days
#101	6th/7th May 1995	Saturday/Sunday
#102	13th/14th May 1995	Saturday/Sunday
#103	20th/21st May 1995	Saturday/Sunday

The particular weekends shown in *Table MW4–7* have a similar pattern to bank holidays. They are a particular period of two days in a particular country—the United Kingdom. They each contain a particular Saturday and a particular Sunday. This pattern of whole–part connections is shown in the space-time map in *Figure MW4–11*. All of the United Kingdom's weekends share the same pattern.

Figure MW4–11 United Kingdom's 6th/7th May weekend space-time map

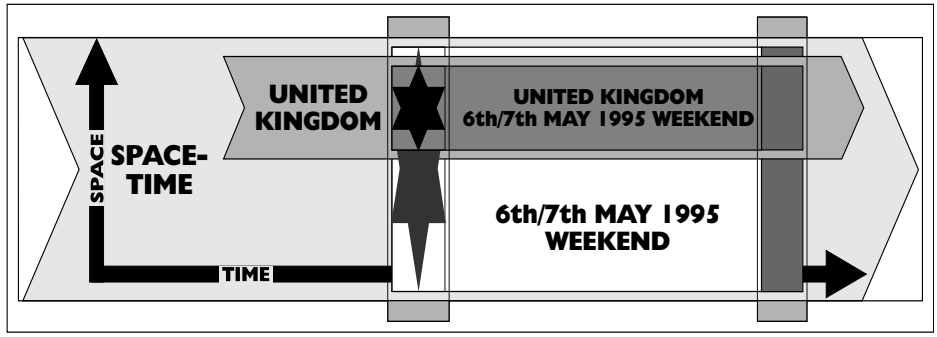
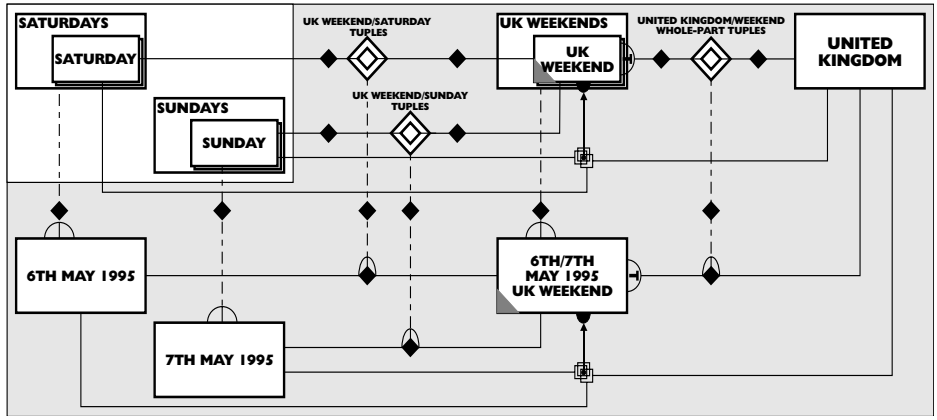
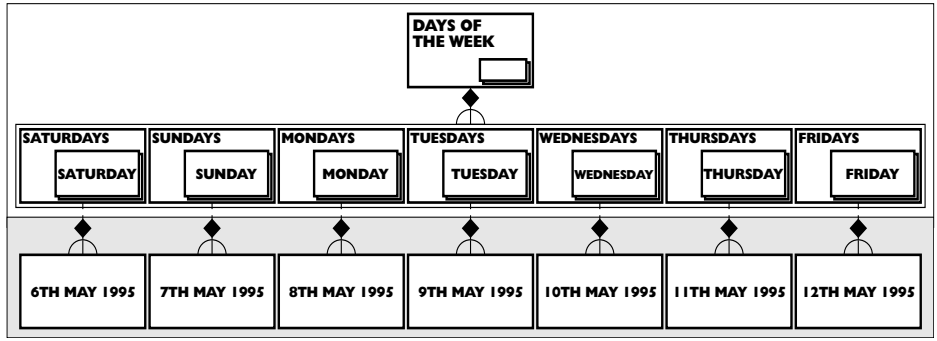


Figure MW4–12 United Kingdom's weekends object schema



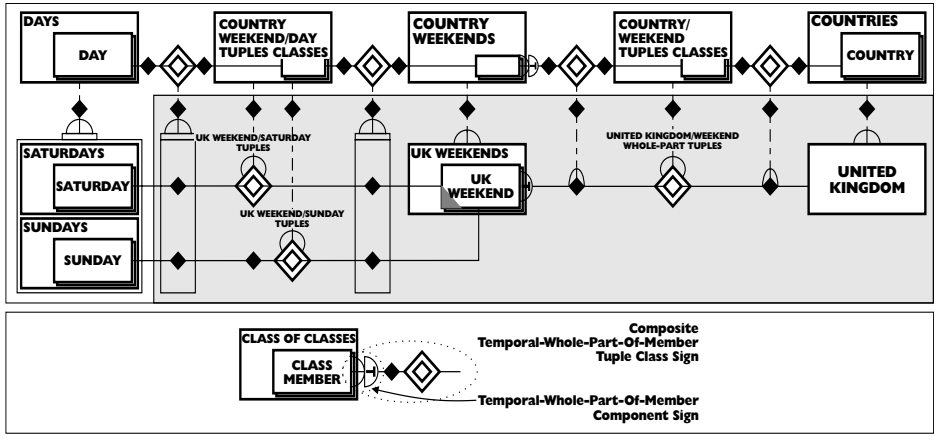
The particular Saturday—the 6th May 1995—belongs to the Saturdays class: the particular Sunday—the 7th May 1995—belongs to the Sundays class. We can generalise this pattern across all the days of the week. This gives us the object schema in *Figure MW4–13*. This is a useful addition to the temporal object model—this general group of objects is common in systems. If we want to, we can also add the sign for a weeks class to the model, where each week is the fusion of seven consecutive days, starting with a Monday.

Figure MW4–13 Days of the week



We are now ready to re-engineer the entity type sign. It is transformed into a class of classes, the country's weekends class, which has, for instance, the U(nited) K(ingdom)'s weekends class as a member. The UK weekends class's tuples are generalised into tuples classes (shown in *Figure MW4–14*). This is similar to the naming tuples' generalisation into naming tuples classes shown in *Figure MW1-33*.

Figure MW4–14 Country weekends class object schema



The generalisation of UK weekends' temporal–whole–part tuples raises a notation issue. Temporal–whole–part tuples link individual objects—not classes. At the UK weekends level, this does not pose a problem because its members are at the individual object level. However, it does pose a notation problem at the level of country weekends, which is a class of classes. Here, there is no individual object to be a whole–part of. This is resolved by using a combination of existing compo-

nents to construct a new composite sign, the temporal–whole–part-member-of sign (shown in *Figure MW4–14*).

5.2 The status of the object model for temporal patterns

This completes the re-engineering of the weekend entity format. It has added two important new groups of patterns to the object model—days and country weekends. These are, like calendar periods, useful general patterns that are found in many systems. Even though the country weekend pattern is lower level, and so less common, it is still useful. It also usefully illustrates how patterns are generalised up the class–member hierarchy as well as the super–sub-class hierarchy.

It is worth noting that a number of the temporal patterns captured in the model, such as particular UK weekends and days of the week, are traditionally implemented as date calculation processes. This is an example of the point that data and process in the information system are not a direct reflection of bodies and events in the real world. Like this case, individual bodies can be implemented as processes.

6 Re-engineering our conceptual patterns for bank holiday and weekend

So far we have constructed the temporal object model by re-engineering entity formats from existing systems. We now look at some of the more complex conceptual patterns that we can re-engineer to make the model substantially more accurate. We look at two patterns:

- Non-country/day holidays, and
- Time zones

All these patterns can be found in manuals listing information on holidays. For example, in the financial sector, SWIFT produces a manual for its customers

whose lists contain these patterns (I have taken the following examples from an old copy).

6.1 Non-country/day holidays

Not all holidays fit into the simple country and day intersection pattern that we have just re-engineered. There are a number of non-country/day holiday patterns. We look at two of them:

- Non-country holidays, and
- Half-day holidays.

Many examples of both of these patterns can be found in the Financial Institution Holidays section of the SWIFT manual. We use some of them to generalise the model.

6.1.1 Non-country holidays

Some holidays apply to only a part of the country. For example, the 9th November 1993 was only a holiday in Madrid—not in the rest of Spain. This gives us an opportunity to generalise the link between countries and bank holidays re-engineered earlier in this paper (illustrated in *Figure MW4–10*).

Madrid is a city and so a geo-political area. We capture the pattern for its 9th November 1993 holiday by generalising the country/bank holiday whole–part tuples link from country to geo-political area. This gives us geo-political area/bank holiday whole–part tuples (shown in *Figure MW4–15*).

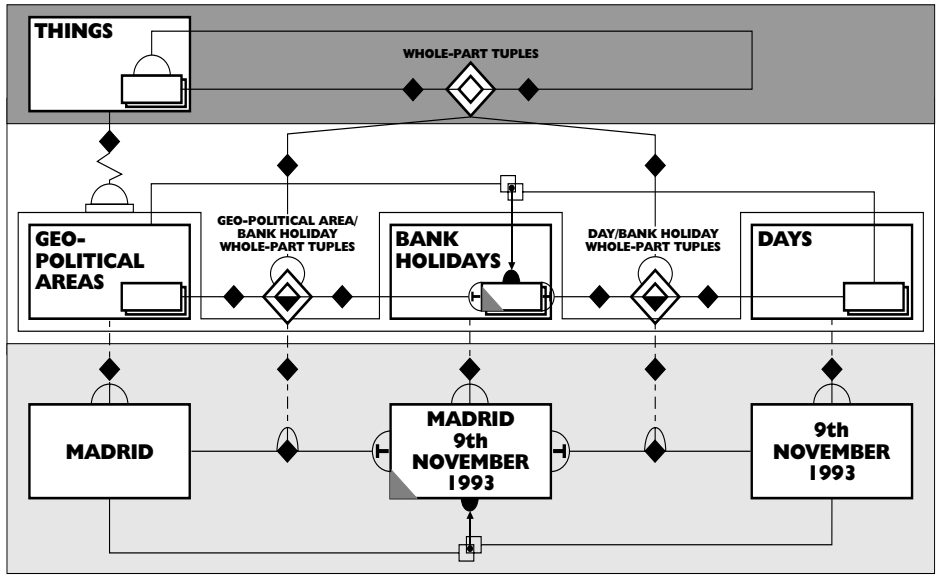


Figure MW4–15 Geopolitical area bank holidays

6.1.2 Half-day holidays

The SWIFT manual has many examples of holidays that last for only part of a day. For example, Korea only has Saturday afternoon as a holiday; people work in the morning. Similarly, Italy, among others, has the afternoons of the 24th and 31st December 1993 as holidays. To re-engineer this, we need objects for mornings and afternoons as both calendar and weekly periods. I leave this as an exercise for you. The object model you produce should generalise bank holidays' connection to day up to a more general calendar period object.

6.2 Time zones

There is one important conceptual pattern we have not touched on yet and that is time zones. We all know that the world is divided into time zones. At a certain time of the year, when it is 2 pm in England, it is 9 am in New York. Different parts of the globe at the 'same' moment in time have different 'times'. Some big countries, such as the United States and Australia, have more than one time zone and

so more than one time. The SWIFT manual has a section that describes these called delta time zones. *Table MW4–8* shows a selection of its information.

Table MW4–8: Selected time zone information

Country	Effective Dates	GMT Differences
Austria	01 Jan 1993 to 27 Mar 1993	+1:00
	28 Mar 1993 to 25 Sep 1993	+2:00
	26 Sep 1993 to 31 Dec 1993	+1:00
Portugal	01 Jan 1993 to 27 Mar 1993	+1:00
	28 Mar 1993 to 25 Sep 1993	+2:00
	26 Sep 1993 to 31 Dec 1993	+1:00
United Kingdom	01 Jan 1993 to 27 Mar 1993	+0:00
	28 Mar 1993 to 23 Oct 1993	+1:00
	24 Oct 1993 to 31 Dec 1993	+0:00

An object-oriented view of *Table MW4–8* sees GMT and local times as objects. The GMT objects are temporal slices straight through the overall space-time object. The local times objects, however, make zigzag slices through space-time, 'overlapping' with different GMT time objects in different countries. For example, the local 2nd January 1993 day starts with the 23:00 GMT event object in Austria and Portugal, but starts with the 24:00 GMT event object in the United Kingdom. This particular zigzag pattern is illustrated in *Figure MW4–16*.

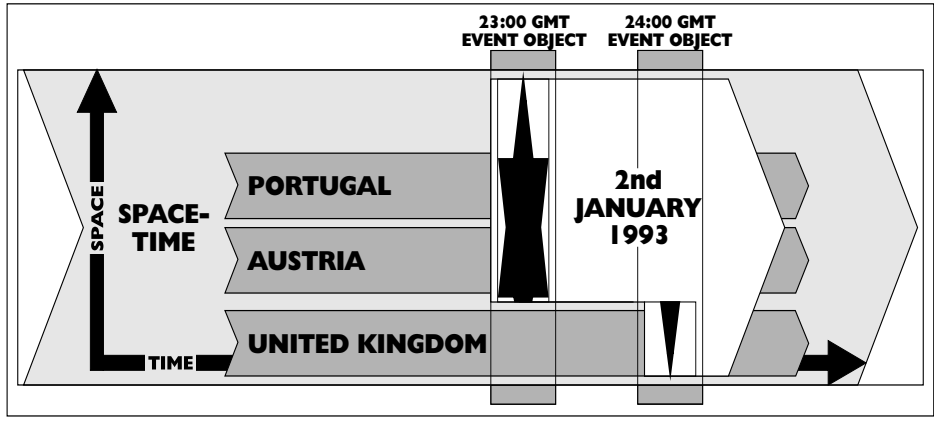


Figure MW4–16 Local day object—a zigzag pattern space-time map

We do not necessarily need to change the model as a result of this analysis. If we are happy with a locally consistent model, then we can say the days class (shown in *Figure MW4–13*) has our local days as members. If, however, we want a globally consistent model, we need to make some changes. We must model both the standard GMT time objects and the various time zones' local times objects. This involves explicitly modelling time zones.

7 The object model for temporal patterns

We have now completed all the work on the temporal patterns object model. The re-engineering of the entity formats for bank holidays and weekends gave us a general model for some simple and basic temporal patterns—calendar periods and days of the week. The brief look at our conceptual patterns has made us aware that existing systems only really capture the simple patterns. There is a range of more sophisticated patterns that we need to model and generalise. However, the model, as it stands, is a good basis for going forward. It is a strong foundation that can be built on and enhanced as more re-engineering is done.

This example also illustrates an important aspect of re-engineering. It shows how the object paradigm's amalgamation of space and time to space-time reveals previously common but mysterious temporal patterns as structural whole–part patterns. Bank holidays are revealed as the intersection of countries and days objects. Country weekends are also revealed to be classes of temporal parts of countries. These examples show how the reference of temporal patterns becomes much clearer when it is revealed as four-dimensional extension. This also clarifies the sense of the patterns; many of the connections between objects are revealed as structural extension-based patterns.

8 Summary

This is the last of the examples. Working through them has achieved a lot. Together they have:

- Given us a feel for what re-engineering is like and the sort of models it produces,
- Shown us the details of a systematic approach to re-engineering the entity formats in the existing system,
- Given us a feel for what analysing object semantics means in practice, and
- Illustrated how the object syntax and notation work on business examples.

In addition they have provided us with three object models:

- Spatial patterns,
- Temporal patterns, and
- Naming patterns.

These contain fundamental and general objects that can be re-used in our re-engineerings.

The worked example papers have focussed on the challenges faced in the process of applying the methodology, rather how a BORO approach affects the way in which a project needs to be managed. *MA1—Starting a Re-Engineering Project* takes up this aspect - and outlines some of the challenges you will face when you set up a re-engineering project using the BORO approach.

8 Summary

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 TIME

A–T

A

C

E

F

O

R

S

T