JOURNAL OF
OBJECT-ORIENTED
Programming

November–December 1994
Vol. 7, No. 7

Cover illustration, Martin Lemelman/SIS

Features

The JOURNAL OF OBJECT-ORIENTED PROGRAMMING (ISSN #0896-8438) is published nine times a year, monthly except for Mar/Apr, Jul/Aug, and Nov/Dec by SIGS Publications Inc., 71 West 23rd Street, 3rd floor, New York, New York 10010. Please direct advertising inquiries to this address. Second class postage paid at New York, New York, and additional mailing offices. POSTMASTER: Send address changes to JOOP, P.O. Box 2030, Langhorne, PA 19047. Inquiries and new subscription orders should also be sent to that address. Annual subscription rates for the U.S. are $199 for institutions, $69 for individuals. All foreign orders must be prepaid in U.S. funds drawn on a U.S. bank. Canadian & Mexican orders add $25 per year and non-North American orders add $40 per air service. For service on current subscriptions, call 215.785.5996, fax 215.785.6073, e-mail p00976@psilink.com.

Manuscripts under review should be typed double spaced (in triplicate) and accompanied by an electronic file in TEXT format. Editorial correspondence and Product News information should be sent to the Editor, Dr. Richard S. Wiener, 135 Rugely Court, Colorado Springs, CO 80906, 719.579.9616 (voice & fax).

Printed in the USA. Canada Post International Publications Mail Product Sales Agreement No. 290343.

Editorial

IT IS THE TIME of year for big and important OOP conferences. The largest of all OOP conferences, the ACM-sponsored OOPSLA '94, is coming up in three weeks. Object Expo Europe took place last week (September 22–26) in London. This was an outstanding conference that brought together speakers and attendees from North America and Europe. For me it provided an opportunity to meet with some JOOP columnists, writers, fellow editors, and friends and meet many European readers. I am told that JOOP is being read in over 70 countries.

As we complete another year of publication I invite our readers to send letters to the Editorial Office or short communications via email (rsweiner@elbert.uccs.edu) that tell us what you like in JOOP and what you would like to see improved. With such a geographically vast readership it is important to receive constructive feedback.

This issue contains four feature articles in addition to our many columns.

"Time invariant virtual member functions dispatching for C++ evolvable classes" by Roger Voss, is a "revisionary supplement to a previous paper entitled, "C++ Evolvable Base Classes Residing in Dynamic Linking Libraries." An evolvable C++ base class is one that can be exported from a shared library and derived from or used in other components. Roger proposes an efficient scheme for implementing a new method dispatching approach.

"On the design of encapsulated CLOS applications" by Charlotte P. Lunau from Aalborg University, Denmark, and Kjeld Larsen from Søren T. Lyngsø A/S Research and Development Centre, examines the encapsulation mechanisms of CLOS and illustrates with a case-study involving a maritime application.

"Modeling the real world: Are classes abstractions or objects?" by Chris Partridge examines the initial stage of object-oriented model construction. It shows how an object-oriented approach provides for "better" models of the real world.

"Object modeling techique: An experience report" by Mohamed E. Fayad, Wei-Tek Tsai, Richard L. Anthony, and Milton L. Fulgum, presents an assessment of the OMT approach to object modeling.

Hope you enjoy this issue of JOOP. Stay warm!

Richard S. Wiener

JOURNAL OF OBJECT-ORIENTED programming

EDITOR
Dr. Richard Wiener
University of Colorado, Colorado Springs

SIGS PUBLICATIONS ADVISORY BOARD
Thomas Atwood, *Object Design*
François Bancilhon, *O_2 Technologies*
Grady Booch, *Rational*
George Bosworth, *Digitalk*
Adele Goldberg, *ParcPlace Systems*
R. Jordan Kreindler, *Rational*
Thomas Love, *Morgan Stanley*
Bertrand Meyer, *ISE*
Meilir Page-Jones, *Wayland Systems*
Cliff Reeves, *IBM*
Dave Thomas, *Object Technology International*

JOOP EDITORIAL BOARD
Daniel Fishman, *Hewlett-Packard Labs*
Stuart Greenfield, *Marist College*
Ivar Jacobson, *Objective Systems*
Boris Magnusson, *Lund University*
Lewis Pinson, *University of Colorado*
Eugene Wang, *Symantec*

COLUMNISTS
Steve Cook, *Object Designers. Ltd.*
John Daniels, *Object Designers. Ltd.*
Desmond D'Souza, *ICON Computing*
Richard Gabriel, *ParcPlace*
Robert Howard, *Tower Technology*
Andrew Koenig, *AT&T Bell Labs*
Wilf LaLonde, *Carleton University*
Mary E.S. Loomis, *Hewlett-Packard*
Stephen Mellor, *Project Technology*
James Odell, *James Odell Associates*
John Pugh, *Carleton University*
James Rumbaugh, *Rational*
Marc Wiener, *Product News Editor*
C. Thomas Wu, *Naval Postgraduate School*

SIGS PUBLICATIONS, INC.
Richard P. Friedman
Founder & Group Publisher

EDITORIAL/PRODUCTION
Kristina Joukhadar, *Managing Editor*
Andrea Cammarata, *Art Director*
Elizabeth A. Upp, *Production Editor*
Margaret Conti, *Advertising Production Coordinator*
Tanya Trowell, *Editorial Assistant*
Andrea Cammarata, *Cover Design*

CIRCULATION
Bruce Shriver, Jr. *Circulation Director*
John R. Wengler, *Circulation Manager*
Kim Maureen Penney, *Circulation Analyst*

ADVERTISING/MARKETING
Shirley Sax, *Director of Sales*
Gary Portie, *Advertising Manager, East Coast/ Canada/ Europe*
Sales Representative, West Coast:
Diane Fuller & Associates 408.255.2991, f: 408.255.2992
Kristine Viksnins, *Advertising Assistant*
Michael Peck, *Advertising Assistant*
Sarah Hamilton, *Director of Promotions and Research*
Caren Polner, *Promotions Graphic Artist*

ADMINISTRATION
Margherita R. Monck, *General Manager*
David Chatterpaul, *Accounting Manager*
James Amenuvor, *Bookkeeper*
Michele Watkins, *Assistant to the Publisher*
Shannon Smith, *Administrative Assistant*

SIGS PUBLICATIONS

Publishers of *Journal of Object-Oriented Programming, Object Magazine, C++ Report, Report on Object Analysis & Design, The Smalltalk Report, ObjektSpektrum (Germany), Objects in Europe (London),* and *The X Journal.*

Chris Partridge

REV-ENG Consulting.
email: 100275.2523@compuserve.com

Modelling the real world: Are classes abstractions or objects?

THERE IS CURRENTLY an imbalance in object-oriented (O-O) systems work. The system building process starts, or at least should start, with a model of the relevant part of the real world, move through various stages, and finish with a working system. However, most O-O systems work is concerned with the later stages of system building, and little time is given to the early stages. We can see this reflected in current O-O literature, which focuses on O-O programming, and often takes current O-O programming languages as a given and considers how to use these to build systems. This is doubly unfortunate as not only is insufficient time spent on considering how to model the real world, but O-O's superior capabilities in this area are not being realised.

This article is an attempt to start redressing the balance. It looks at the initial stage of O-O system building—modelling the real world—taking only the notion of an object as a given. It shows how an object-oriented approach at this stage enables us to build better models of the real world and how this, in turn, clarifies a fundamental aspect of the O-O paradigm—revealing its expressive power.

A simple example—types of cars in a business system—is used for illustration. An implementation model of car types is looked at and found wanting—it does not reflect the real world properly. A clearer understanding of car types is built up by going back to fundamentals and examining our concept of a type of object—the class object. This is done firstly by a brief look at the nature of the object revolution, then by contrasting the pre-object abstraction paradigm for concepts with the object-oriented paradigm's "objectification" approach—and comparing both of these with one derived from object-oriented programming (OOP). This clearer understanding of class objects is then applied to the car types example, giving a simple picture of what car types (and objects like car types) are in the real world. Hopefully this illustrates the value of using an O-O approach to modelling the real world.

USER-DEFINED TYPES

Let's consider our particular example. We are familiar with user-defined types such as account, invoice, and deal types. Without them users would not have the option of setting up their own types; they would have to use the ones hard-coded into the system. The use of these type objects is commonplace in business systems, whether O-O or traditional; almost every system of a reasonable size has a number of them. System builders are so familiar with them that often they can—with little or no analysis—intuitively work out how to implement the type objects required. The ease and familiarity of the implementation of type objects means that obvious, but awkward, questions that would arise during real-world modelling do not get asked such as: what are type objects and what do they model in the real world? An object-oriented approach enables us to give useful and interesting answers to these questions.

Car types

Let's look at a simple example of a system with user-defined types: car types. Let's say that in our system there are cars and car types. Cars can be saloon cars* (i.e. the class Cars has a subclass Saloon Cars). Car types can be saloon car types (i.e., the class Car Types has a subclass Saloon Car Types). Each car has to be of a particular type (i.e., each instance of the class Cars is a type of an instance of the class Car Types; with each instance of the class Saloon Cars restricted to being a type of an instance of the class Saloon Car Types). There is one type of car already set up in the system (i.e., there is one instance of the class Saloon Car Types)—Premier. There are two saloon cars set up in the system (i.e., instances of the class Saloon Car)—Car # 123 and Car # 456, both of which are Premiers. This is diagrammed in Figure 1.

The type structure in this model should be familiar to most system builders. Now, ask the obvious but awkward question: what does the model tell us about what car types are and what they refer to in the real world? Not a lot; this is because it is an implementation model—i.e., a model of how the system is to be implemented—and so reflects the computer system, not the real world. The model is not the result of describing the real world (the first stage in the system building process), but a description of the system to be implemented. If we are going to follow the con-

* "Saloon car" is British English for sedan car.

ventional system building process and start by building a real-world model then we need to understand what car types (and types in general) really are. Most people have not felt the need to acquire this understanding, so we need to build it up now.

BACKGROUND

Building up this understanding can take some effort, not because the ideas are complex—they are simple—but because they are new and different. It is best achieved by examining the meaning and, in particular, the history of the class object: this is a bigger subject than is generally realised.

Notion of an object

Origin. In the 4th century BC, the ancient Greek thinker Aristotle introduced the notions of *entity* and *attribute* as the fundamental information particles in his information paradigm. Although there have been some rival theories since, these notions are still in use today—after over two thousand years. They have had a remarkable staying power.

Only recently has the object paradigm emerged to challenge it. Gottlob Frege (1848–1925), in his work published in the latter part of the 19th century, made the breakthrough that started the shift to the object paradigm rolling. I outline below how he introduced the notions of an object and a concept defined in a logical (rather than psychological) way; how he recognised that classes were objects and general concepts referred to class objects. Although this technical jargon may sound daunting, the underlying ideas are really simple—simpler and less confused than the ideas they replace—as will become clear as they are explained.

Object conceptual revolution. Frege's breakthrough started a conceptual revolution that is still going on today. It is establishing an information paradigm based upon objects as the fundamental information particles. The revolution began in information science and is now alive and kicking in the computing community, where it is known as object-oriented computing.

Fundamental particles. What are fundamental information particles? We build our conceptual structures out of basic conceptual building blocks—fundamental information particles. As our understanding progresses we go through conceptual revolutions, replacing the old types of fundamental information particles with new types.

In physics, a similar type of revolution has occurred frequently—this century physicists have changed their types of fundamental *physical* particles many times. Physicists started the 20th century with the atom as the fundamental physical particle. Everything from aardvarks to zebras were made out of atoms. When they divided the atom, they shifted to new fundamental particles; electrons, neutrons, and protons. They then divided these particles and shifted to a profusion of new types, such as leptons and fermions. In their latest paradigm, they have shifted to a single type of fundamental particle, superstrings.

The difference between the fundamental building blocks that physics and information science deal with is that physics deals with *physical* building blocks, whereas information science deals with *conceptual* building blocks.

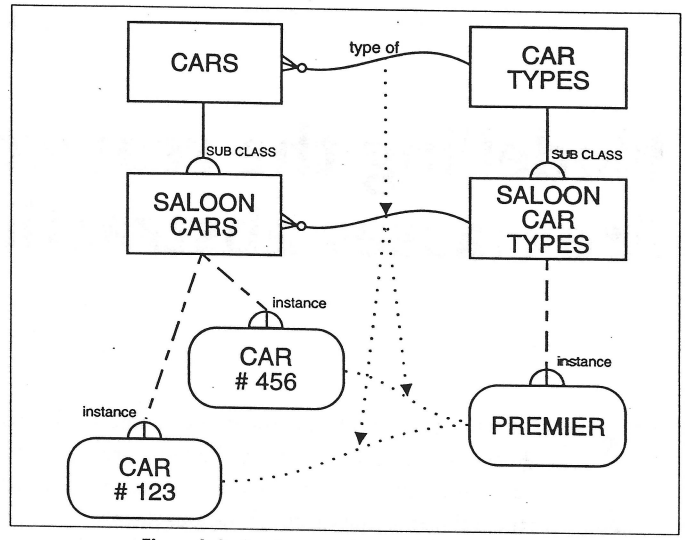

Figure I. An implementation model of car types.

Clearer relationship with real world. In computing, one of the perceived benefits of the O-O approach is the ease and elegance with which it models the real world—the objects in an O-O computer system are more likely to be reflections of real-world objects than is true of other approaches.

This links to a perceived benefit of the object approach in information science, which has the major task of explaining how concepts relate to the real world. It uses the object paradigm, which neatly explains how general concepts in a notation refer to the real world—they refer to class objects. The importance of this achievement is shown by comparing the abstraction paradigm's approach to concepts—which has no explanation of how general concepts refer to the real world—with the object paradigm's objectification approach.

Comparing paradigms for concepts

The abstraction and object paradigms for concepts are described and compared below. They are then compared with a paradigm for concepts derived from OOP. As you read on, bear in mind that the abstraction paradigm is an attempt to answer the question of how we learn concepts and use them to make judgements, whereas the object paradigm answers the question we raised above; to what do general concepts refer?

Abstraction. In current usage, the word "abstraction" has acquired many meanings. Historically it had a definite meaning, referring to a particular mental process. We put abstractionism into its historical context, capturing the definite meaning; describe how it works; and then look at examples.

Origin of abstraction. Abstraction is a mental mechanism for creating ideas. The notion of abstraction has been around for a long time. The person normally credited with introducing it is the English thinker John Locke (1632–1704), although Aristotle (384–322 BC) mentions it in passing. John Locke gave abstraction a central place in

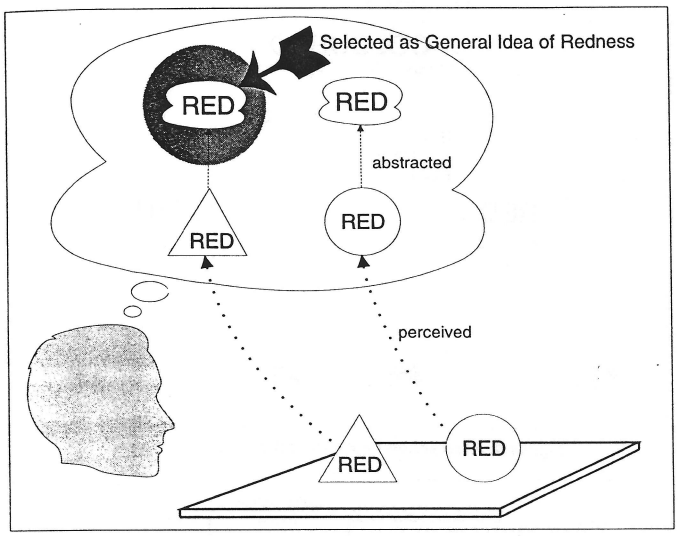

Figure 2. Schema of me learning my general idea of redness.

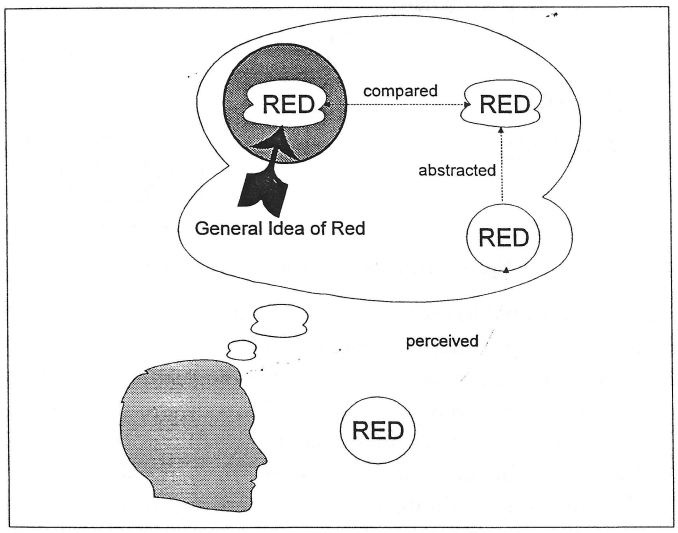

Figure 3. Schema of me using my general idea of redness.

his thinking, claiming it was the process we use to stock our minds with *general ideas* (Locke's name for general concepts). Although his abstraction paradigm may now seem awkward and inadequate, we should remember that we are looking at it with the benefit of centuries of hindsight—at the time it was a real achievement. We should also remember that people still use it, almost unchanged, today—for example, look at the description of a logical model as an abstraction of a physical model in a system modelling textbook.

What is abstraction? A key element of abstraction is the mental process of taking an idea of a particular object and generating a general idea by focusing on a single property of that thing and ignoring all its other properties. John Locke describes this process and gives us an example:

> . . . the Mind makes the particular Ideas, received from particular objects, to become general; which is done by considering them as they are in the Mind such appearances, separate from all other Existences, and the circumstances of real Existence, as time, place, or any other concomitant Ideas. This is called ABSTRACTION, whereby Ideas taken from particular Beings, become general representatives of all the same kind; and their Names general Names, applicable to whatever exists conformable to such abstract Ideas. . . . Thus the same colour being observed to day in Chalk or Snow, which the Mind yesterday received from Milk, it considers that appearance alone, makes it a representative of all that kind; and giving it the name Whiteness . . . and thus Universals, whether Ideas or Terms, are made.[1]

There are levels of abstraction; a general idea can be made more general by mentally ignoring some of its properties. Frege (who thought abstractionism nonsense) gives us a good description and example:

> Since everything is an idea, we can easily alter objects by directing our attention towards this and away from that. The latter is particularly effective. We take less notice of a property, and it vanishes. By causing one characteristic after another to vanish we attain to ever more abstract objects. Concepts too, are therefore ideas, merely less complete ones than objects; they have only those properties from which we have not yet abstracted. . . . Let us suppose, for example, there are sitting side by side in front of us a black and a white cat. We pay no attention to their colour: they become colourless, but are still sitting side by side. We pay no attention to their posture: they are no longer sitting, without, however, assuming a different posture; but each is still in the same position. We cease to attend to their places: they become devoid of position, but continue to be apart from one another. We have thus, perhaps, attained from them a general concept of a cat.[2]

Frege continued, writing that by repeated application of the operation of abstraction, "every object is transformed into an ever more bloodless ghost" and, in abstraction, "the objects are essentially altered thereby, so that the objects brought under the same concept became more similar to one another."

We can see that there are levels of abstraction, and these levels form a hierarchy of abstractness; with ideas being more or less abstract versions of other ideas.

Let's consider an example (the "red example," which will be used again later) of how we learn about general ideas and use them to make judgements. I have two red shapes (a triangle and a circle) on the desk in front of me. I form an idea in my mind of each object, creating two ideas in my mind, one for each shape. I now ignore all the characteristics of these ideas except their redness, creating two more ideas of similar appearance; I pick one of these as the representative idea and call this a general idea of redness (see Fig. 2). This illustrates how we learn about general ideas using abstraction.

Now I look at another red shape, and form an idea of it—still remembering the general idea of redness. I now ignore all the characteristics of my idea of the new red shape except its colour—I end up with a general idea that has the same appearance as my general idea of redness; so I deduce that the new shape has redness, or is red (see Fig. 3).

This illustrates how I use abstraction to make judgements as to whether a particular idea falls under a general idea; it is rather like bird spotters comparing a particular bird with pictures in a book to identify its species.

We can expand this example to "coloured": I also have two green shapes on my desk; I can use them to abstract to the idea of greenness. I can use any or all of the shapes to abstract to the more

general idea of coloured. I now have a hierarchy of abstractness (shown in Fig. 4).

Let's consider conceptual structure. The diagram in Figure 5 (called a meta object schema) shows the underlying conceptual structure of abstraction. Notice that the two relations "abstracted from" and "more abstract than" are both created by the process of abstraction. We will compare this with the meta object schemas of the other conceptual structures.

Issues. This abstraction paradigm is at heart psychological. Considering the psychological problems inherent in the paradigm gives us an insight into how it works.† In Locke's time, psychology had not evolved from philosophy and it was not standard practice to verify philosophy by experiment. While the paradigm may give a plausible account of how a person uses general ideas, there is little experimental data to verify it.

You can see this by doing an experiment; go through the red shape example observing yourself. Are you aware of yourself making the idea of a shape in your mind or abstracting the colour of a shape? Do you recognise the new shape by abstracting the particular idea and comparing it with some general abstract idea? While your lack of awareness of the abstraction mechanism does not prove it does not exist—it may be working at an unconscious level—it also does not prove it exists.

Abstraction is not the only way to acquire general concepts; we can acquire a general idea without encountering particular examples. For instance, botanists often learn about a species of flower from illustrations and descriptions before encountering an example. In these circumstances there must be something other than abstraction at work.

The paradigm is also representational; it assumes that ideas are some sort of picture of what is being represented. We often talk in a representational way, saying things such as "I have a picture of it in my mind." The red triangle and circle icons in Figure 2 are also representational; they have a similar look to the things to which they refer. The impulse to link ideas to the real world by some relation of similar appearance is a good one, but unfortunately it does not work. There are severe limitations, particularly for more "abstract" general ideas. Locke was aware of this: he offers this example;

> For example, does it not require some pains and skill to form the general idea of a triangle, (which is yet none of the most abstract, comprehensive, and difficult) for it must be neither Oblique, nor Rectangle, neither Equilateral, Equicrural, nor Scalenon; but all and none of these at once. In effect it is something imperfect, that cannot exist; an Idea wherein some parts of several different and inconsistent Ideas are put together.[3]

We can see the same problem in the awkward representation of abstract redness in Figure 2: the general idea of redness has no shape or size, but the icon representing it has a definite shape and a definite size. It is not only a problem forming a representational general idea, but, once formed, often the idea cannot refer to anything in the real world, because there is nothing in the real world that can be similar in appearance. For instance, the general idea

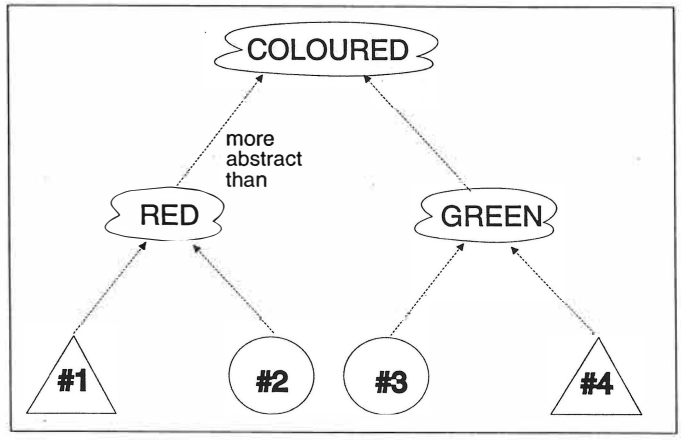

Figure 4. Abstraction hierarchy schema of "coloured."

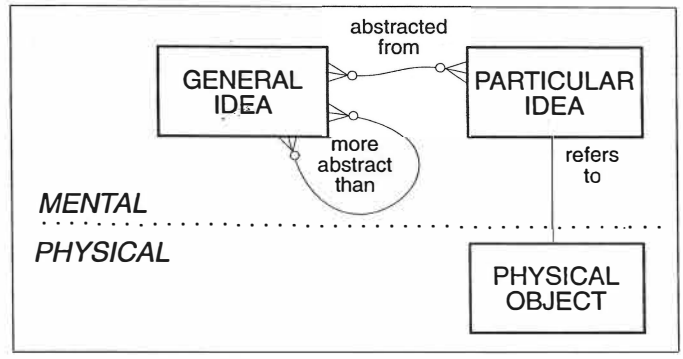

Figure 5. Meta object schema of abstraction.

of redness would have to refer to something in the real world with no size or shape—clearly an impossibility.

Our real concern is not with the psychological problems described above. Psychology studies the working of the human mind—which we can consider as an implemented system, albeit implemented in biological technology. Abstraction, if it does exist, is a feature of human thought; it is not meant to and obviously does not apply to computer system processing. Our interest here is in the general characteristics of systems, not one particular type of system; our interest is in what a system does, rather than in how a particular implementation does it.

These are logical questions. The particular logical question that concerns us is how ideas relate (i.e. refer) to the real world. An idea or concept of an individual physical object refers to that object; this is not problematical, the relation is shown in Figure 5. But to what does a general idea refer? Under an abstractionist paradigm it does not refer at all, it is a mental entity, part of a mechanism that we use to recognise properties of things. It is quite clear from the meta object schema (Fig. 5) that for an abstractionist a general (mental) idea is not directly related to anything in the real world; it does not refer to anything in the real world. So this paradigm does not address our question at all.

Objectification. As with abstractionism, it is useful to put objectification into its historical context and describe how it works, before giving examples. By Frege's time (the late 19th century), psychology had evolved out of philosophy into a separate discipline, but the philosophy of concepts, information science, was still rooted in abstractionist psychology. Frege saw the problems, both psychological

† Peter Geach in his book MENTAL ACTS (1957) provides a detailed criticism of the psychological aspects of abstractionism.

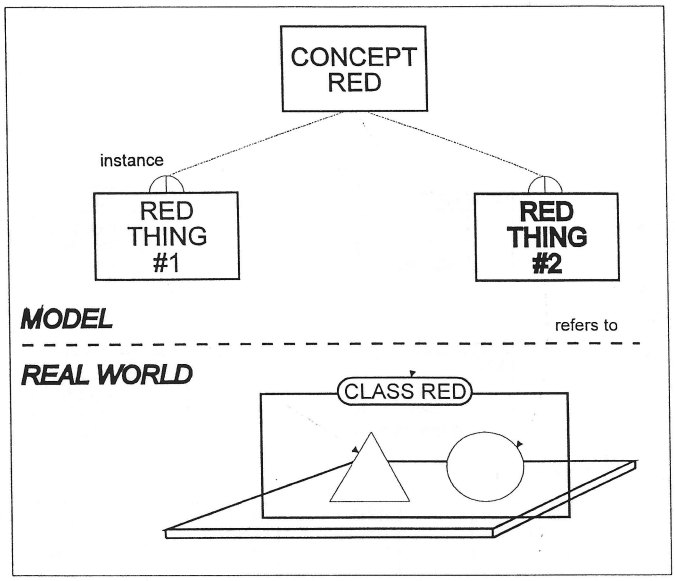

Figure 6. Schema of "red."

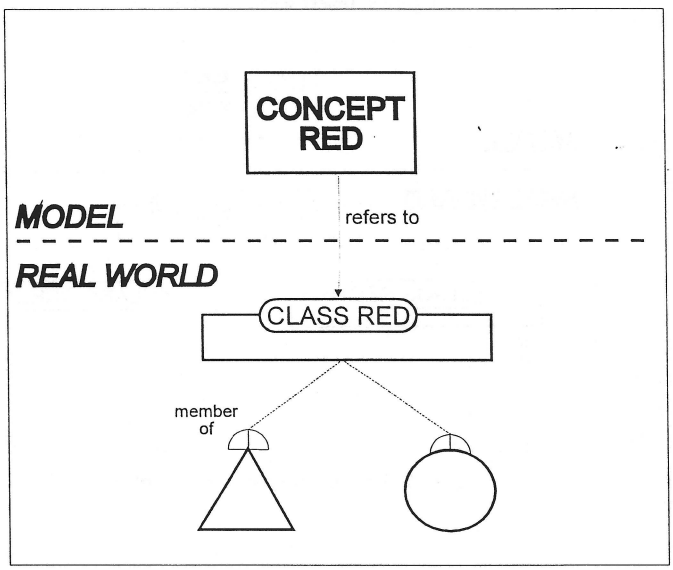

Figure 7. Reference diagram for the concept red.

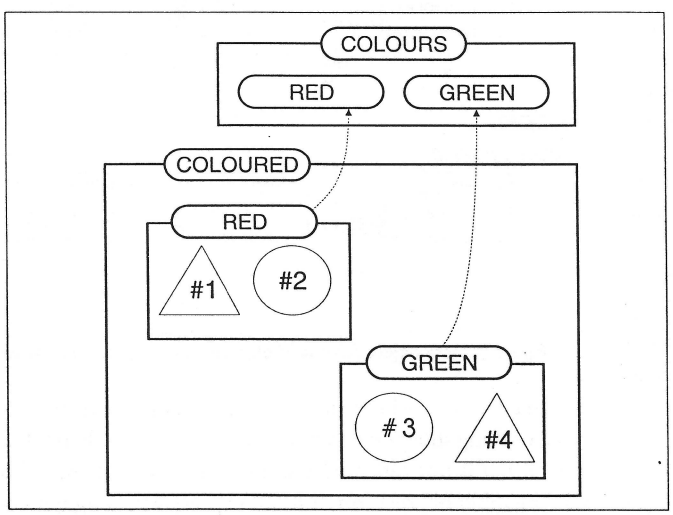

Figure 8. Real-world schema of classes colours **and** coloured.

and logical, with abstraction. He proposed a new paradigm for concepts with a logical rather than a psychological basis.‡ Unlike Locke's abstractionism, Frege's paradigm is not meant to deal with psychological matters. It deals with logical matters, which Locke's abstractionism signally fails to do. Under Frege's approach, a general concept (the logical name for Locke's psychological general idea) is the name for a class of objects.§ This answers the question—to what does a general concept refer? A general concept refers to a class of objects.

This much should be familiar to O-O practitioners. In this scheme of things the abstraction hierarchy is replaced with a super–subclass hierarchy. Where abstractionists say this general concept A is more abstract than that concept B, Frege would say that this general concept A refers to a subclass of the class referred to by that concept B.

However Frege could and did go a step further. He suggested a class of objects was also an object in its own right. In recognising the class as an object, one is objectifying it—hence the name *objectification.* A shift like this from one paradigm to another opens up new avenues of thought. This is the case here. If we acquire general concepts by abstraction, and make them more general by further abstraction, then there is only an abstraction hierarchy. Things are different if classes are objects. They can be grouped together to form a class—a class of classes. Classes of classes are also objects and so can also be grouped together into classes—classes of classes of classes—and so on, ad infinitum. Treating classes as objects opens up a whole new hierarchy, the class-member hierarchy.

This means general concepts refer to something physical, something in the real world. Physical objects obviously belong to the real world, so each class of physical objects and class of classes of physical objects (and so on) does as well: unlike Locke's general ideas, Frege's general concepts refer to classes (class objects) that belong to the real world. In the same way that names of physical objects (singular concepts) refer to an object, a general concept is a name that refers to a class object. This gives us a simple and clear relationship between concepts (general or singular) and the real world.

The practical import of this can be made clear with an example. Let us reuse the example of red shapes. I start again with the two red shapes on the table. I classify the shapes as red, constructing the class red. Figure 6 shows that I now have three objects in the real world: two physical objects and one class object.

Notice how the structure of the model reflects the structure of the real world. Each of the objects in the real world has a corresponding object in the model—the relationship between the concept red and the real world is shown in the reference diagram in Figure 7.

We can now expand the example to include classes of classes. Let's consider the two green shapes and classify them as green things—constructing the class green. I now have two classes, red and green. In a similar way to the abstractionist example, I can now classify all the red and green shapes as coloured shapes and construct the class coloured. Unlike the abstractionist example, I can also classify the two class objects, red and green, as colours—con-

‡ Frege recognised the importance of this, making it a "fundamental principle . . . always to separate sharply the psychological from the logical."

§ This is a simplification of Frege's position, he looked upon concepts as "unsaturated entities." The terms used here reflect contemporary usage rather than being an historically accurate reflection of Frege's usage.

structing the class colours. In this situation, represented in Figure 8, red and green are both types of colour. Notice abstractionists cannot make this move because it cannot be done by abstracting.

Figure 8 shows that the classes red and green are both subclasses of the class coloured things and members of the class colours. The form of the diagram in Figure 8 is not wholly satisfactory: a class object (such as red) that is also a member of another class has to be iconised twice—once as a rounded rectangle *and* a rectangle and again as just a rounded rectangle.

A more satisfactory way of showing this is to diagram the concepts that refer to the objects as in the object schema shown in Figure 9.

Using this approach, each concept is only iconised once. Each class icon contains a members icon; this enables the diagram to distinguish relations between classes (such as "subclass") from relations between members (such as "instance of" in the next object schema—Fig. 11). It also ensures that a distinction is made between the name of the class and the name for its members: there is a tendency in O-O computing to call a class by the name of its members.

The relationship between concepts and the real world is *reference*. The reference diagram in Figure 10 shows how reference works for concepts that refer to classes of classes.

Solution. Frege rightly recognised that a general concept refers to a class of objects. This effectively answers the question of what a car type is—it is the class of cars of that type. The objectification paradigm is a neat solution to the logical problem of the reference of general concepts—which was not solved by abstractionism. His insight that classes were objects and so should be treated like objects and be "allowed" to be members of classes significantly enhanced the expressive power of the paradigm, enabling it to describe things that could not be described under abstractionism.

The meta object schema in Figure 11 shows the underlying conceptual structure of the objectification paradigm.

Compare this with the meta schema for abstraction (Fig. 5). There are two main points to note. First, in the objectification schema the model structure reflects (one could say *simulates*) the real-world structure; this ensures that the model is a reflection of the real world. There is no feeling of reflection in the abstraction schema. Second, the objectification modelling notation is more powerful and can produce a more expressive model of the real world. The real world portions of these two schemas show the types of thing in the real world that can be described using that modelling notation: they illustrate the notation's power. Comparing the two schemas, it is obvious that the objectification modelling notation is much stronger, in that it can be used to describe a whole new hierarchy: the class-member hierarchy.

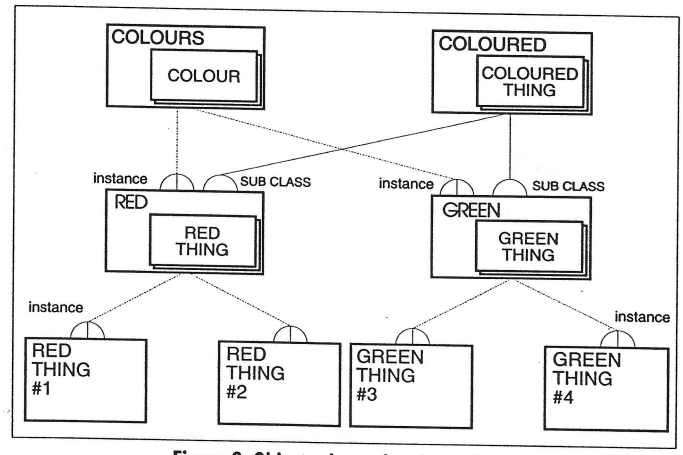

Figure 9. Object schema for class colour.

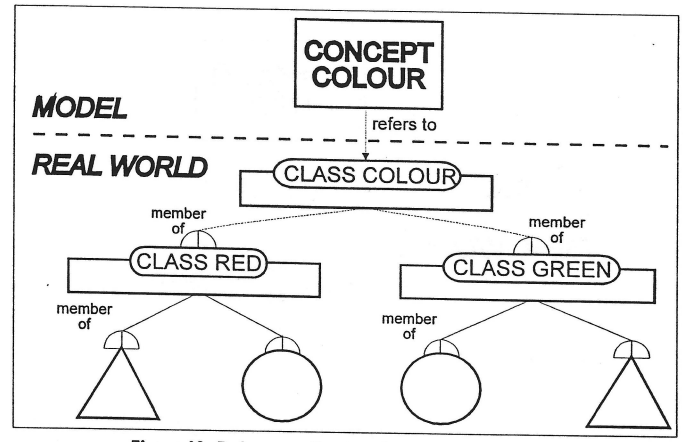

Figure 10. Reference diagram for concept "colour."

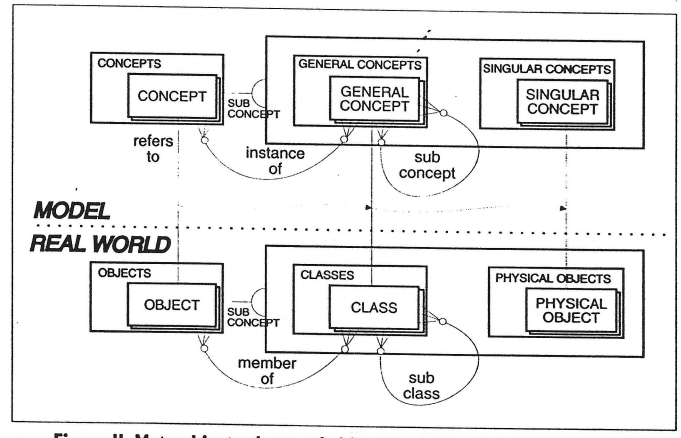

Figure 11. Meta object schema of objectification conceptual structure.

OOP-BASED CONCEPTUAL STRUCTURE

Even though OOP does not have an explicit paradigm for concepts, we can derive (or, more technically, reverse-engineer) one. The conceptual structure of the OOP-based paradigm for concepts is shown in the meta object schema depicted in Figure 12.

Compare this with the meta schemas for abstraction (Fig. 5) nd objectification (Fig. 11). This schema is similar in structure to the objectification schema. The model reflects the real world. The real-world portion of this schema is the same as the objectification schema because we know that we need be able to describe at least the types of thing shown there. However, there is one element of the real world that is not reflected accurately in the OOP-based model—the "member of" relation. It is not expressive enough; it cannot describe classes higher up the class-member hierarchy than classes of physical objects. The "instance of" relation is similar in structure to abstraction's "abstracted from" relation (see Fig. 5); we can see that the OOP- based paradigm has an abstractionist legacy.

The OOP-based paradigm could be called a classification paradigm for concepts as it recognises classes, but even though it

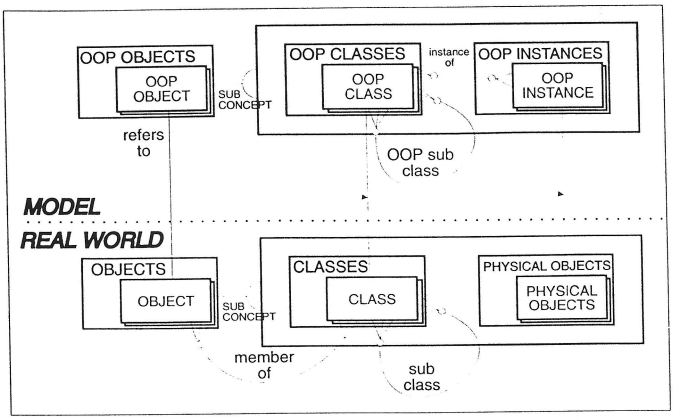

Figure 12. Meta object schema of OOP-based concept paradigm.

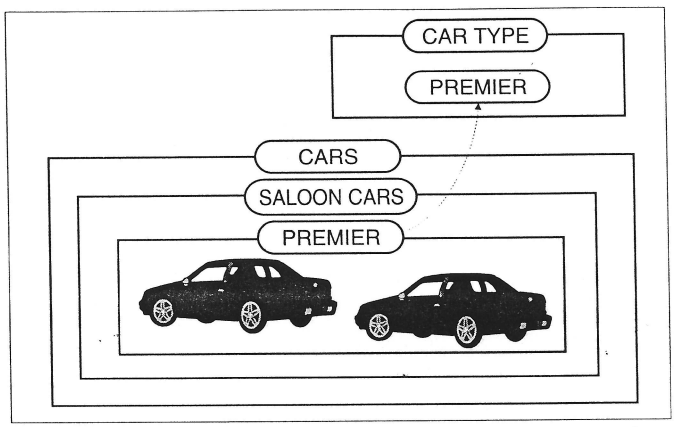

Figure 13. Schema of car types.

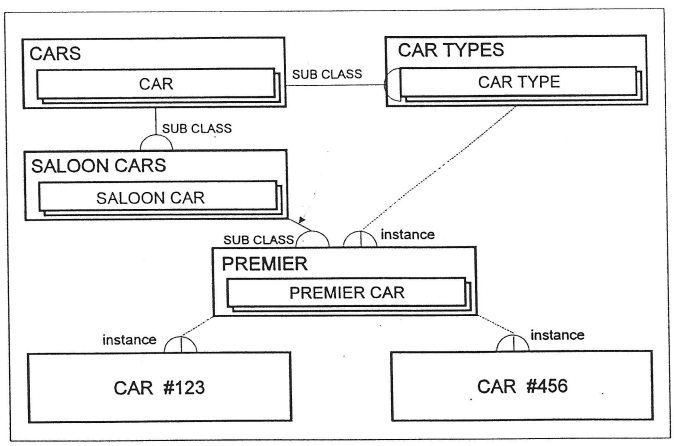

Figure 14. Object schema for real-world model of car types.

calls them objects, it does not give them the full rights of objects. Enhancing a classification paradigm with the increased power of the objectification paradigm does not increase its complexity, it only changes its shape. Only a small change is required; changing the "instance of" relation between OOP Class and Instance to one between OOP Class and OOP Object.

I suspect that there is a conceptual barrier that stops a lot of O-O practitioners (and so OOP) from going further than the classification paradigm and making the shift to objectification. I think it is because they are still under the influence of the abstraction paradigm; they still think of general concepts in terms of abstraction. This is hardly surprising as abstractionism is deeply embedded in the way most people think. When eventually its influence is eroded, and the object paradigm becomes the natural way to think, the classification paradigm will disappear too.

REAL-WORLD MODEL OF CAR TYPES

Now let's build a real-world model of our car types example. Look again at Figure 1, and also at the real-world schema in Figure 13.

Ask yourself what Premier really is. Premier is a type and so it is a class, the class of Premier Cars. It is also a member of the class Car Types; so Car Types is a class of classes.

If we were to build the model using a classification paradigm for concepts then we would be faced with an awkward dilemma: if we recognise Premier as a class then our paradigm is not expressive enough to also recognise it as a member of the class Car Types; similarly, if we recognise it as a member of the class Car Types, we cannot recognise it as a class. System builders need to make a choice between these two options, neither of which is satisfactory. They tend to choose the second, which is probably the better of the two, but leaves us with the problem of explaining what types are in the real world. Considering these problems, it is understandable that we do not spend time asking what a car type really is.

However, if we have an objectification paradigm for concepts it does make sense to ask the question. We can model the real world directly recognising Premier as a class and a member of the class Car Types. The object schema for the real-world model is given in Figure 14.

Compare the real-world and implementation models (in Figs. 1 and 14). Just by looking at the two models, one can tell that the real-world model is simpler in structure; comparing the number of relations the real model has six to the implementation model's nine. Premier is now a class and also a subclass of the class Saloon Cars; Cars #123 and #456 are members of the class Premier. The real-world model is more expressive, making clear the nature of car types. Our understanding of what the concepts really refer to give us a semantically richer perception of what is going on. The model gives a simpler, clearer (and more accurate) reflection of what is actually going on in the real world.

SUMMARY

We have seen that we can get a better understanding of the nature of things and a simpler model when we look closely at what the objects in our systems actually refer to in the real world. This better understanding and simpler model are only possible, in this case through the use of an O-O approach. We have also seen how focusing on the real world can give us a better understanding of the O-O paradigm. Hopefully this will persuade you to spend more time modelling the real world. ■

References

1. Locke, J. An Essay Concerning Human Understanding, Book II, Chap XI, Sect. 9, Clarendon Edition, Oxford University Press, Oxford, 1975 (originally published in 1689).
2. Frege, G. Review of Husserl's 'Philosophie der Arithmetic' ZPK, vol CII 312-313, 1894. Translated by Kluge, E., Mind, LXXXI, 321-337, 1972.
3. Locke, J. An Essay Concerning Human Understanding, Book IV, Chap VII, Sect. 9, Clarendon Edition, Oxford University Press, Oxford, 197 (originally published in 1689).