Object-oriented Concepts for Software Development and Conceptual Modelling in GIS and Surveying

Walter Hesse PhD (Melbourne)
Major Projects Group
Melbourne Information Technology Services (MITS)

and

William Hazleton, PhD (Melbourne)

and

Ian P. Williamson PhD (NSW), LS, MIS(Aust)
Professor of Surveying and Land Information
Department of Surveying and Land Information
The University of Melbourne

Introduction

Recent years have seen a growing research interest amongst Information Technology professionals for Object Oriented Programming Systems (OOPS) and Object Oriented Geographic Information Systems (OOGIS). The nature of this research for the 'classical' surveyor demands a move away from the 'comfort zone' of Surveying Techniques, Mapping and LIS/GIS into the more fundamental areas of Computing Science. This move is essential for a better understanding of the necessary data modelling and general object-oriented concepts for Information Systems in general, and LIS/GIS more specifically. The aim of this paper is to give an overview of this relatively new and specialised field for the Surveying professional interested in GIS issues.

Software Quality and Change - The Enemy

Brad J. Cox, in his book "Object Oriented Programming: An evolutionary Approach" [Cox 1987], gives an alarming account of software costs versus results in nine U.S. Federal software projects from a 1979 U.S. Government Accounting Office Report, shown in figure 1:

![Figure 1 - Software Costs versus Results](image)

The breakdown of the total cost shows that 47% were paid for but never delivered, 29% were delivered but never used, 19% were abandoned or reworked, 3% were used after change, and less than 2% were used as delivered. (Data from ACM Sigsoft Software Engineering Notes, vol 10 no 5, Oct 1985).

Adding to the initial costs of software purchase are often considerable software maintenance costs. Bertrand Meyer, in his book "Object-Oriented Software Construction" [Meyer 1988] gives a breakdown of these costs, based on a survey of 487 installations developing various application software (shown in figure 2):
The high percentage of changing user requirements is not surprising in view of the rapid technology changes and increasing user expectations. The high costs involved reflect on conventional software and its inability to cope with change.

This situation can reach crisis proportions where ad-hoc fixes or extensions to existing modules become increasingly difficult to manage and 'change becomes the enemy' [Cox 1987]. In a later section on object-oriented languages and database systems, the example of the Apple Macintosh system software shows that often a radical breach with tradition and a complete re-write of the software is required in order to cope with the change.

The change in data formats becomes a major concern, especially when not controlled early in the development stages of a new proposed system. On examining the GIS situation in Australasia it is clear that with the notable exception of New Zealand, very few attempts have been made by authoritative sources to set standards for street address, land appellation or general data interchange. Meyer gives an example from the US, where the post office changed to a "5+4" postal code (i.e. large companies could use nine instead of the standard five digits) and subsequently a large number of application software had to re-written to cope with the change, resulting in a considerable expense.

Software quality aspects can be grouped into internal and external quality factors [Meyer 1988]. The internal quality factors apply to the "inner workings" of software modules such as modularity or readability, and are generally of not much concern to the end user of the software, unless the user is concerned with self-developed extensions. Of more importance for the end user are external factors such as:

**Correctness** the ability of software products to exactly perform their tasks, as defined by the requirements and specifications.

**Robustness** the ability of software systems to function even in abnormal conditions.

**Extendibility** the ease with which software products may be adapted to changes of specifications.

**Reusability** the ability of software products to be reused, in whole or in part, for new applications.

**Compatibility** the ease with which software products may be combined with others.

Meyer argues that the quality of all these factors will be enhanced by the use of object-oriented design. His book covers the advantages of object-oriented design as well as other aspects of software engineering, and he demonstrates these with his own language development, EIFFEL. This leads to a complete new re-thinking of conventional software creation and maintenance concepts; Object Orientation.

**General object-oriented Concepts**

An interesting comparison between the evolution of hardware and software technology is presented by Cox [1987] and shown in figure 3.

Where recent decades of information technology development have seen an exponential growth in hardware technology, only arithmetic growth has been achieved in software technology.
While hardware engineers were learning to build with modules whose size and power were increasing exponentially, software engineers continued to build programs just as always, by choosing programming statements one by one, putting them side by side to make a program...

The key to the issue lies in the degree of modularity that can be achieved with software components and the questions whether application software should be "data" or "process" oriented. An ideal situation is to devise a software system that is build out of small components, which Cox calls Software-IC's similar in concept to that of hardware Integrated Circuits (IC's). These components would have a well defined behaviour and the ability to be shared amongst a whole variety of different applications so that the building of very complex systems would be reduced to a simple gathering task of all required components.

These concepts can be achieved in a similar way through structured programming and the division of an application into a series of subroutines or function calls. However, object-oriented programming and data modelling goes beyond this approach, as outlined in the following section. The object oriented language Smalltalk is used as an example.

**Specific Example: Smalltalk/V286 and WIDGETS**

One specific example of an object-oriented language, in form of the Smalltalk/V286 implementation by Digitalk, Inc. [Digitalk 1988], is used here to further explain the concepts of the approach. This and the Smalltalk/Windows version [Digitalk 1991] are used for the case study of a cadastral maintenance problem which is described in [Hesse and Williamson 1992].

Digitalk's implementation is both a powerful language and a very good program development environment. It runs on Macintosh and IBM compatible personal computers. The Windows version has full access to Dynamic Link Libraries (DLL) and Dynamic Data Exchange (DDE), so programs written in other (conventional) languages can be easily incorporated and data can be shared with foreign applications.

Program development is based on "Smalltalk's Big Ideas" [Digitalk 1991]:

- **The most important component in a computing system is the individual human user.**
- **Programming should be a natural extension of thinking.**
- **Programming should be a dynamic, evolutionary process consistent with the model of human learning activity.**
- **A computing environment is both a language and a productivity enhancing interface of programmer/user "power tools" - utilities to express yourself in that language and to organise and flexibly use both procedural and factual knowledge.**

Some of these objectives or ideas require explanation. The language is easy to learn because of its simple syntax and semantics. Basic concepts such as `object`, `class`, `message` and `method` form the basis of Smalltalk programming:

**Objects** are self-describing data structures and every object is an instance of a `class`. The class defines the structure and behaviour of all its instances. The data stored inside an object is accessible only through messages. Thus objects are safe and protected data structures. Smalltalk variables are containers for objects. They can be either private (accessible only to a single object, noted as lower-case) or shared (accessible to multiple objects, noted as upper-case).

**Classes** describe data structures (objects), algorithms (methods) and external interfaces (message protocols) and could therefore be viewed as program modules in conventional languages. The *encapsulation* of code and data, or information hiding, is a big advantage in itself because it encourages a
high level of modularity and safe software components. What makes object classes really different is the
concept of **class hierarchy** and **inheritance**. On top of this class hierarchy is a root class, called Object, all
other classes are subclasses of Object or its subclasses. Class Object, as a so called superclass, provides
common behaviour for all its subclasses, such as methods for testing an objects class or for copying an
object. Each subclass builds on its superclasses by adding its own methods and instance variables to its
behaviour. It is possible to override the inherited methods of a superclass by re-defining the methods at
subclass level. The concepts of inheritance allow for easy extension of software capability and code re-
use.

This is illustrated by the example of an animal hierarchy, taken from *Digitalk [1991]* and shown in figure
4:

![Figure 4 - Animal Class Hierarchy](image)

Here the class Animal is subclass of class Object, Bird and Mammal are subclasses of class Animal, etc...
The instance variables defined for this example could look like:

Animal (name, knowledge, habitat, topSpeed, colour, picture)

Bird (flying)

Parrot (vocabulary)

Penguin ( )

Mammal ( )

Dog (barksAlot)

Whale ( )

Through the mechanism of inheritance an object combines the "knowledge" (instance variables) of its
superclasses with its own instance variables, resulting in:

Parrot

(name, knowledge, habitat, topSpeed, colour, picture, flying, vocabulary)

Penguin

(name, knowledge, habitat, topSpeed, colour, picture, flying)

Dog

(name, knowledge, habitat, topSpeed, colour, picture, barksAlot)

Whale

(name, knowledge, habitat, topSpeed, colour, picture)

Sending **messages** to objects activates processing in Smalltalk (see figure 5). The message determines
which object's **method(s)** get activated. The beginning of a method defines its name, arguments and any
temporary variables that may be needed.
Figure 5 - Communicating Objects

The obvious difficulty of 'flying penguins' is overcome by re-defining the inherited "flying" method at Penguin level, typically with a message response such as: "I can't fly!". All other Bird knowledge is perfectly valid and is inherited without the need for further re-definition.

Different objects, which are not necessarily in the same branch of the class hierarchy, can define the same method, but with quite different result For example the method "printString" may be defined differently for object class Integer, Array or Character, so

423 printString.

#(1 2 3) printString.

'This is a string' printString.

will result in the representation of all objects on paper, through exactly the same message. The capability of different objects responding individually, with their own unique behaviour, to the same message, is known as polymorphism. Here it relieves the programmer from remembering unique "print" vocabulary for each class.

The recommended development cycle for Smalltalk, given in Digitalk [1991] as:

- State the Problem
- Draw the Window
- Identify the Classes
- Describe Object States
- List the Object Interfaces
- Implement the Methods

puts emphasis on the graphical user interface ("draw the window" - "look-and-feel") very early in the development stages of a new project. Modularity during development is enforced by a "divide-and-conquer" approach of defining single object (classes) and their interface to other objects through message passing. At no stage does a complete and final overview of the desired application have to be "designed", such as complex data flow diagrams in conventional language programming efforts. The programmer is only concerned with the behaviour of self-contained units, called objects. An object is simply a related piece of code and data encapsulated in a "black box". This allows a natural modelling of reality, where objects rely on the capabilities of other objects to perform certain specialised tasks. Just like people trust a motor mechanic to fix their cars, no knowledge of the inner workings of a mechanical repair shop, their component supplier or knowledge of the qualifications of the mechanic is required to achieve the result of a repaired car.

Although Smalltalk is an interpreted language, and therefore relatively slow for certain numeric operations (for example heavy 'number-crunching' such as matrix manipulations) it has the big advantage of very fast prototyping, without the need for tedious re-compilation of source code. Every change in code is interpreted as soon as a Smalltalk expression is evaluated. The user does not need to worry about memory management, such as memory allocation or de-allocation. This is automatically taken care of by a so called 'garbage collector' mechanism within the Smalltalk environment.

Comparison with Conventional Programming Languages

The usual sequence of software creation with conventional languages is shown in figure 6. One or more
source code subroutines, in this example FORTRAN, have to be compiled into machine instruction versions of the subroutines or "modules" (to avoid the confusing term "object" files), with .OBJ file extensions. These modules are then linked with other system resources, such as input/output routines and peripheral control, into a single executable program (with .EXE file extension).

Figure 6 - Conventional Software Creation

After a desired source code change, the source code always has to be re-compiled and re-linked again. This can become a very tedious task during the development and testing stages, although utilities such as "make-file" utilities (which keeps track of file changes based on the "last-edited" file attribute and link dependencies to other source and system files) and "debugger" (for step-wise execution and variable inspection) are supplied with most modern conventional languages.

Of more difficulty is the tracking of changes that are made due to a change in the data model of the application and the reflection of this change through the various subroutines. For example, a simple CAD system may be designed for a number of graphic primitives (lines, arcs, circles, blocks ...) and a number of subroutines for their manipulation (create, delete, scale, move, rotate, change colour ...). Every subroutine must have some form of decision branching mechanism, for example somewhere in the general "move" routine the following branching has to occur:

...IF (element.eq.'line') THEN
CALL move_line (parameter_list)
ELSE IF (element.eq.'arc') THEN
CALL move_arc (parameter_list)
ELSE IF (element.eq.'circle') THEN
CALL move_circle (parameter_list)
ELSE IF (element.eq.'block') THEN....
ENDIF...

These program branches would have to be included in a similar fashion in each general element manipulation subroutine. This becomes increasingly difficult to manage, should either a new element (such as a "b-spline") or added functionality (such as "change line style") be desired for a new version of this example CAD package. The grouping of similar behaviour of two (or more) graphic primitives into the same subroutine could be achieved with:

...ELSE IF (element.eq.'arc'.or.
element.eq.'circle' ) THEN
CALL move_arc_circle (parameter_list)...

This approach will become entirely confusing, should specific functionality require a grouping of different graphic primitives. One could expect considerable difficulty in trying to implement a much more sophisticated GIS, with all its additional data input, management, analysis and data representation capabilities, with this approach. In an object-oriented approach, the very nature of the object class hierarchy allows for inheritance of functionality and the management of possible change within self-
contained objects or appropriate subclassing, should for example a new element with its own new and specific behaviour be required.

Digitalk's Smalltalk version is delivered with over one hundred and sixty different object-classes with over three thousand methods. The system is completely open, except for some low-level assembler calls, and the users are encouraged to create their own specialisations through sub-classing of objects, or to go on "extensive cut-and-paste raids through the existing class hierarchy" to create their own new objects.

Other Object-Oriented Languages and Database Implementations

A survey conducted during the 1987 OOPSLA Conference [OOPSLA'87 Editor 1987] asked the question: "What OOPS system(s) or language(s) do you use?"

![Figure 7 - OOPSLA'87 Survey](image)

The graph has been created from the results of the original survey, supplied in list form of 413 responses. The language variations, based on the same original have been combined into groups as shown in the graph legend. Note that "C based" includes C++, Inheritance C, Objective C, etc. The language variations based on Smalltalk still make up the largest group, followed by the "C based" group of which 60% use C++.

The fore-runner language Simula, hidden in figure 7 amongst "other", was designed in 1967 by Ole-Johan Dahl and Krysten Nygaard from the University of Oslo and the Norwegian Computing Center and is in fact an object-oriented extension of Algol 60. Simula 67, as it used to be called until 1986, was originally developed for event simulation but has since matured into a general-purpose programming language maintained by the "Simula Standards Group". Simula has since 1967 sparked the development of a number of other more successful object-oriented attempts or, as Meyer [1988] puts it in the preface to his book:

... born in the ice-blue waters of the festooned Norwegian coast; amplified (by an aberration of world currents, for which marine geographers have yet to find a suitable explanation) along the much grayer range of the Californian Pacific; viewed by some as a typhoon, by some as a tsunami, and by some as a storm in a teacup - a tidal wave is reaching the shores of the computing world. "Object-oriented" is the latest IN term, complementing or perhaps even replacing "structured" as the high-tech version of "good"...

Similar thoughts regarding the "flavour of the month" idea, are expressed by King [1989]:

... It's interesting to note, however, that there is considerable disagreement concerning the definition of "object-oriented". We know it's a good thing, but not everyone agrees on what it is ...

Amongst the "real" object-oriented languages are the C Extensions C++ and Objective C, although Meyer, as mentor of his own object-oriented language EIFFEL, calls them "hybrid" and

... as in the FORTRAN case, the emulation is only obtained by doing violence to the language ... The danger in trying to force object-oriented concepts onto a C base is to get an inconsistent construction, impairing the software development process and the quality of the resulting products. A hybrid approach yields hybrid quality...

Also of note are the Lisp and Prolog extensions amongst the artificial intelligence community (such as Loops, developed at Xerox and Flavors, developed at MIT) and certain object-oriented features of Ada,
commissioned by the U.S. Department of Defence (DOD).

For the popular Apple Macintosh platform of computers, Object PASCAL - a successor to the Clascal language used to develop some of the Lisa and Macintosh software - is available, for more detail see Schmucker [1986]. The importance of object-oriented concepts for the Apple Macintosh environment is highlighted by Meng [1990]. He points out the object-oriented tradition of the Mac with its HyperCard and HyperTalk programs included with the standard system delivery. The new Macintosh system 7.0 ('Finder'), for example, is entirely (re-written) in object-oriented code. Meng claims that Steve Jobbs original ideas for the creation of the Apple Macintosh were inspired by the Smalltalk language and GUI.

No current (1991) data could be obtained on the usage of object-oriented languages across the various hardware platforms, but the general feeling from discussions and the browsing of computing magazines suggest a strong growth in recent years of C++ in its various vendor versions. Australian Macworld [1989] reviews C++ as the leading language and reports, that many Macintosh based software modules are currently completely re-written in this language (for example, the 4th Dimension database package) which marks a clear move away from the traditional Mac language, PASCAL.

The more traditional programming languages will certainly not become obsolete in the immediate future. Lieberherr et al [1988] discusses general object-oriented style and the fact that still many improvements could be applied to the technology. Alabiso [1988] argues strongly for a possible and useful reconciliation of structured analysis (SA) and object-oriented (OO) design, a view shared by Nicholas Wirth, the inventor of PASCAL [Borland 1991]. Yourdon [1990] discusses the current transition trend in software development and the general software technology evolution and points out, that

...just as there will always be a job somewhere on the planet for renegade assembly language programmers, there will always be a home for those who want to draw data-flow diagrams....

The very nature of object-oriented design (objects as 'carriers' of both data and procedures) combines the conceptual model, language and database model into Object-Oriented Database Management Systems (OODBMS). This is a fundamental conceptual difference to that of merely conventional programming languages, which are only concerned with procedures (for example 'read', 'manipulate' and 'write') or their 'handling' of data. Ochuodo [1990] explores the limitations of existing conventional data base models, such as network, hierarchical, relational and semantic models. He is particularly critical about their inability to be extended in their functionality and explains the emerging knowledge-based representations and object-oriented models. Dobbie [1990] gives a thorough survey of a number of current OODBMS, such as O2, Iris, ORION, POSTGRES, Gemstone, O-Raid, Zeitgeist, PROBE, Cactis, VBASE and ONTOS.

The differences between the currently most popular relational data base model and the emerging object-oriented data base model is discussed in more detail in the following section.

The Relational vs the Object-oriented Data Model

The comparison between the two models can be made under a number of different aspects. Lee [1991] examines the areas of efficiency, data semantics, model extensions, object identity and programming interface; and finds several weak points of the relational model in these areas. There are obviously a number of advantages in the relational data base model, compared to earlier data base modelling efforts, such as the hierarchical and the network data models (see also Date [1980]). These are, according to Smith and Zdonik [1987]:

...The relational model is more flexible and easier to use than previous database models. It is more flexible because inter-record relationships do not have to be predefined. The relational join operator allows a user to relate records dynamically based on attribute values. The relational model is easier to use because of its more intuitive metaphor of tabular data...
The discussion over the years on the advantages of the new object-oriented database design over the relational data base design amongst computer scientists has been fierce at a very fundamental level. Stone and Hentchel [1990] in their paper entitled "Database Wars Revisited" found:

...When all the dust settled from the great database wars of the 1970s, Codd's 12 rules for a relational data model stood triumphant over the hierarchical and network database. In the 1980s, however, came a challenge called the "Object-Oriented Database Manifesto", written by a gaggle of notable academics specialising in object-oriented technology. The relational camp's response - "Third Generation Database System Manifesto" - was not slow in coming. At stake, after all, were the hearts, minds, and purchasing budgets of database designers and users everywhere...

This discussion and the necessary experimentation on a more application oriented level, such as GIS, has taken place only gradually and at a less frenetic pace. Surveyors, map-makers and GIS implementors seem to be cautious of adopting the new ideas on a large scale. They tend to sit back and wait for commercially proven solutions before a change is even considered - Land Information New Zealand being the refreshing and courageous exception to the rule within Australasia [LINZnews 1989].

Ed Yourdon [Bouldin 1989] points out that the introduction of any new software technique will always take up a considerable amount of time, and asks:

... Why does it take so long? Because 80% of the staff is working on maintenance projects and has no opportunity to use new technologies of any kind. Because the rest of the staff is working on a "crunch mode" project and they're too far behind schedule to take the time to learn a new method of developing systems. Because the person who introduced the new technology has little or no political clout and even less communication skills. Because nobody has presented senior management with any convincing evidence about the economics of the new technology ...

There is clearly more to the establishment of a better system than just technicalities and the necessary training. The political aspects can not be underestimated. Quite often the advocates and implementors of an 'old' technology are established in the management decision process and fear the introduction of something 'new' that will supersede their earlier efforts and will take something out of their control.

The measurement of the success of a new technology is often reduced to efficiency benchmark testing where for example the speed of data access in data base search operations is measured. Duhl and Damon [1988] write about a performance comparison of object and relational databases using the "Sun Benchmark". They note the current lack of other meaningful performance benchmarks for this comparison and use VBASE from Ontologic Inc. (see Dobbie [1990] for more detail), for the comparison with earlier relational test results. They conclude that

...the results offer strong evidence that object databases are capable of performing as well as, or better than, existing relational database systems...

...object databases will provide higher performance for several reasons: Operations can be performed on individual objects or classes of objects; sub-components can refer to an object by object identity rather than by state(key); Object references can be cached for in-memory access times; and complex design components can be represented more directly using objects than with relational systems...

Even if the speed-of-retrieval argument can be won (here quite comfortably) in favour of a new technology, quite often the user of the traditional technology is reluctant to change, presenting the argument "I don't care if it takes longer - we get faster hardware next year, anyway!". It is much harder to judge the advantages of a richer data model and its impact on better general modelling of a particular user
situation [Duhl and Damon 1988]:

...One of the major attractions of object systems is the high conceptual level and abstraction at which users can approach, interact with, and model their problem domains...

Smith and Zdonik [1987] compare relational and object-oriented database systems in their hypermedia case study. They find that for this particular application, the use of the relational database system becomes awkward, as the information is made up of complex, hierarchical data structures that have to be 'flattened' into relations. Hodges [1989] finds that

...The advantages of OODBMSs is that they move closer to the real world, managing larger and larger pieces of unstructured data...most of the data in the world is unstructured...

For more details on the sometimes difficult, but absolutely necessary, general 'normalisation' process of relational table structures see for example Date [1980]. This difficulty in the Australasian LIS context is documented by Firns and Benwell [1991]. Their Entity-Relationship modelling [Chen 1976] efforts for the case of a local authority responsible for issuing building permits, conclude with

...if the relationships are not correctly reflected in the ER model, then the resulting database schema will almost inevitably lack the flexibility to support the production of any information other than that for which it was originally designed, thus reducing the value of the data...

The advantages of 'relations' as a general modelling tool, or 'semantic constructs', are described by Rumbaugh [1987] and found to be not very well supported by object-oriented programming and while possible to implement it is;

...not possible to separate the abstraction from the implementation with the same clarity as found in the relational model...

Rumbaugh finally presents the object-relation model, which combines the concepts of objects, classes, and methods from the object-oriented model with the concept of relations from the entity-relationship model and describes it as useful, practical and natural.

In summing up the discussion about the general differences between the two models, it becomes apparent that the transition from the old RDBMS to the new OODBMS does not have to be radical. Certain similarities could allow a smooth transition from one into the other and make OODBMS appear as natural (!) extensions of the relational model, and not as a threat of totally conflicting ideas.

The special nature of GIS in its application for spatial problems seems to 'tip the scale' in favour of the object-oriented approach for this particular application, see also Egenhofer and Frank [1988c and 1989]. See Calkins and Marble [1987] for their attempts to design an automated cartographic production tool using the relational database model.

Object-oriented concepts in GIS

In the more application oriented field of LIS/GIS, the United States National Centre for Geographic Information and Analysis (NCGIA) has had a number of scientists working on projects which are already making use of OOPS concepts. These include the NCGIA research initiatives 'Languages of Spatial Relations', 'Multiple Representations' and 'Advanced Systems for Cartographic Design' [Morgan 1989].

These initiatives have particular relevance for this review and include, or overlap with, the problem areas of general object-oriented data modelling, cadastral data base maintenance applications and cartographic applications such as better data quality visualisation. References to these fields are explained in the following sections.
General Data Modelling

For a full appreciation of OOGIS concepts it was important to study examples of current conventional technology. Cowen [1988] includes a clear definition of the differences between CAD, DBMS and GIS. A similar approach is taken by Sutherland and McDonald [1989] who describe the various data models currently used for spatial information systems.

The difficulties in extending the CAD functionality to incorporate GIS analysis and image processing is best described in Logan and Bryant [1987]. Similar attempts to expand CAD are found in Ramirez [1991a] who proposes a Universal Computer-Aided Mapping System (UCAMS) with increased user-friendliness.

More modern GIS versions, based on the very popular relational data model, are described by Abel [1988]. Chang and Kao [1991] propose extensions to this model in the form of a semantic data model. The fundamental need for the right conceptual spatial data model is discussed by Lee [1991] and Lee and Isdale [1991]. Oosterom and Bos [1989] and Oosterom and Vijnbrief [1991] give an insight into object-oriented approaches to the design of GIS.

Dominant in the area of general data modelling for OOGIS, especially the design of object-oriented query languages for GIS, is the work at the Department of Surveying Engineering at the University of Maine, U.S. Different aspects of OOGIS are covered by their work, for example Egenhofer and Frank [1988a] on object-oriented human interface aspects; Egenhofer and Frank [1988b] on PANDA, an extensible DBMS; Egenhofer and Frank [1988c] on object-oriented modelling; Palmer and Frank [1988] on spatial languages; and Bruegger and Frank [1990] on hierarchical extensions of topological data structures.

An important source and description of a conceptual data model for a modern GIS is presented by Guptill [1991] in the form of the USGS Enhanced Digital Line Graph (DLG-E). This proposal provides for an interesting comparison with the German ATKIS proposal, described by the AdV [1988 and 1989] and Hesse and Lealhy [1990] and [Hesse and Williamson 1992]. On the U.S. Spatial Data Transfer Standard see for example Moellering [1987], McDermott [1991] and ALIC [1990].

Cadastral Applications

The concepts of object-oriented techniques for cadastral applications such as 'shifting cadastre' and an actual example using HyperCard, are given by Hebblethwaite [1989]. Mullin [1988] examines the issues concerned with various forms of digital cadastral data updates. The research of Kühn [1989 and 1990] on the construction and editing of general geometric constraints, can be applied to cadastral data base maintenance problems. The work of Kjerne and Dueker [1988] is notable as they propose the modelling of cadastral spatial relationships using Smalltalk-80. The use of an object-oriented database for adjustments in a measurement-based multi-purpose cadastre is described by Buyong and Frank [1989].

Current shortcomings of Australasian cadastral systems and the need for object-oriented research towards better cadastral maintenance procedures is documented by Hesse and Williamson [1990] and Hesse et al [1990]. Background information on similar problems experienced in German cadastral systems can be found in Hesse [1986]. A prototype solution for some of the problems is given by Hesse et al. [1990].

Cartographic Applications

A good reference on object-oriented design for cartographic applications was found in Steiner et al. [1989]. Here an object-oriented package is proposed to overcome the shortcomings of conventional procedural languages for cartographic systems, especially concerning the variation of map symbology.

This work can be categorised into the wider research field of multiple representations. See for example Oaten [1990] and Oaten and Shortis [1990a,b] on the necessary concepts and implementation issues. These multiple representation issues are important for the creation of cartographic expert systems and the
inherent problems of map generalisation. Müller [1990] investigates rule based generalisation and describes its potential and impediments. He proposes the creation of 'knowledge tanks' for rule based generalisation [Müller 1991].

Map generalisation will be a major research topic for future advanced cartographic GIS output; a local Australian example is found in Brooke et al. [1991]. Its complexity will require advanced software techniques and data structures. This will have a strong bearing on the necessary conceptual GIS data model and spatial data transfer standards (see also [Hesse and Williamson 1993b]).

Conclusions

The purpose of this paper was, to give a general introduction into the potential and benefits of object-oriented programming systems and their underlying data models. Their historic background, general concepts, the various emerging languages and database implementations, a specific Smalltalk example and a comparison with conventional programming languages was presented. The comparison of the popular relational and the object-oriented data model is made possible due to the nature of the object-oriented concept, which extends it from a pure programming language to a much wider and more powerful data modelling and storage mechanism. As a consequence of this review a number of conclusions can be drawn regarding the application of OOPS for surveying and GIS applications, as follows:

a. Conventional software maintenance issues are a major and costly concern for the suppliers and users of large, complex systems.
b. These issues can reach crisis proportion should changing user requirements demand amendment. Difficulties caused by changes in data formats make the definition of the appropriate standards very important in the early stages of any (geographic) information system.
c. A number of general software quality aspects, such as correctness, robustness, extendibility and reusability can be identified, in order to combat the difficult task of building software systems. These aspects become increasingly difficult to control with conventional programming languages and lead to the development of object-oriented languages and data base models.
d. The key for success seems to be in the areas of software re-usability, in the form of software IC's, called objects and classes. This approach could ensure the change from current arithmetic growth in software technology to future exponential growth, similar to the ongoing evolution of hardware technology.
e. Smalltalk was one of the first attempts in the 1970's to achieve this goal and recent years have seen a strong increase in demand for object-oriented languages on a variety of different hardware platforms. C++ seems to becoming the most successful implementation of the concept and many conventional programming languages are beginning to allow for object-oriented extensions.
f. The conceptual model, the language and the database model can be combined in Object-Oriented Database Management Systems (OODBMS). A number of OODBMS have recently become commercially available, competing against the traditional and highly popular relational database systems. Despite a number of advantages, the introduction to the market place and widespread use is slow, mainly due to the nature of any new information technology acceptance.
g. OODBMS should be viewed as a powerful extension to RDBMS, not as a radical or opposing threat. They differ in three main ways from their relational predecessors: encapsulation of data and processes within the data; inheritance of existing class behaviour through the means of subclassing; and polymorphism, the characteristic of having different objects responding to the same message.

References

Abel, D.J. 1988, "SIRO-DBMS A Data Base Tool-Kit for Geographical Information Systems", Information Technology Technical Report, CSIRO, Division of Information Technology, North Ryde, NSW
AdV 1988, Amtliches Topographisch - Kartographisches Informationssystem (ATKIS) - Das Vorhaben der Landesvermessungsverwaltungen zum Aufbau Digitaler Landschaftsmodelle und Digitaler Kartographischer Modelle, "Gelbes Heft", Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder (AdV), Bonn

AdV Arbeitsgruppe ATKIS (Editor) 1989, Amtliches Topographisch - Kartographisches Informationssystem (ATKIS), Gesamtdokumentation, Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder (AdV)


Australian Land Information Council (Editor) (ALIC) 1990, National Strategy on Land Information Management, Australian Land Information Council

Australian Macworld 1989, "Special Focus - C++ The leading Language", Australian Macworld, 110 - 111


Bouldin, 1989


Buyong and Frank, 1989


Cowen, D.J. 1988, "GIS versus CAD versus DBMS: What are the Differences ?", Photogrammetric Engineering and Remote Sensing, Vol. 54, No. 11, ASPRS, Falls Church, VA.


Egenhofer, M.J., Frank, A.U. 1988b, "PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques", GIS/LIS '88 Workshop, Department of Surveying Engineering, University of Maine

Egenhofer, M.J., Frank, A.U. 1988c, "Object-Oriented Modelling: A Powerful Tool for GIS", GIS/LIS '88 Workshop, Department of Surveying Engineering, University of Maine


Elliot, B. 1988, "Considerations in buying a Geographic Information System", Unpublished paper, WILD, Denver, Colorado


Herring, J.R., Larsen, R.C., Shivakumar, J. 1988, "Extensions to the SQL Query language to support
Hesse, 1986


Hesse, W., Williamson, I.P. 1990


Hesse, W., Williamson, I.P. 1993a, "ATKIS"

Hesse, W., Williamson, I.P. 1993b, "Transfer Standards"


Hodges, P. 1989, "A Relational Successor ?", DATAMATION, November 1989, 47-50


Intergraph 1987, "Object oriented programming and applications benefits", Internal document, Intergraph Corporation, Huntsville, Alabama


King, R. 1989, "My Cat is Object-Oriented", Object-Oriented Concepts, Databases, and Applications, 23-30, ACM-Press, Addison-Wesley

Kjerne, D., Duecker, K.J. 1988, "Modelling Cadastral Spatial Relationships using Smalltalk80", GIS/LIS '88, American Society for Photogrammetry and Remote Sensing, Conference Proceedings, 373 - 385, ASPRS, Falls Church, VA

Kühn, W. 1989, "Interaktion mit raumbezogenen Informationssystemen", Mitteilungen Nr. 44, Institut für Geodäsie und Photogrammetrie an der Eidgenössischen Technischen Hochschule Zürich


Lee Y.C.1991, "A Comparison of Relational and Object-Oriented Models for Spatial Data", Department of Surveying Engineering, University of New Brunswick

Lee Y.C., Isdale M. 1991, "The Need for a Spatial Data Model", Department of Surveying Engineering, University of New Brunswick


LINZnews 1989, "TIGRIS Arrives", LINZnews, No. 9, 6-7, DOSLI, Wellington, New Zealand


Meng, B. 1990, "Object-Oriented Programming", MACWORLD, January, 174 - 180


Morgan, P.S. 1989, "NCGIA Update", NCGIA Update, Vol. 1, No. 2, National Center for Geographic Information and Analysis

Müller J.C. 1990, "Rule Based Generalisation: Potentials and Impediments", Proceedings of the 4th International Symposium on Spatial Data Handling, 317-334, Department of Geography, University of Zürich


Mullin, R.C. 1988, "Data Update in a Land Information System", UNISURV S-34, The University of New South Wales, Kensington, NSW


Oaten, G.W., Shortis, M.R. 1990a, "Multiple Representations for GIS", SIRC Colloquium Proceedings, SIRC, University of Otago

Oaten, G.W., Shortis, M.R. 1990b, "Geometric Elements or Thematic Features: The Quest for Basic Spatial Objects", AURISA 90

Ochuodo, S.J.1990, "Object Oriented Database Support for Software Project Management Environments", Department of Computer Science, University of York


Oosterom, P. van, Vijlbrief, T. 1991, "Building a GIS on top of the open DBMS Postgres", EGIS'91 Proceedings, April 2-5, 775-787


Schmucker, K. 1986, Object-oriented Languages on the MacIntosh, Apple Press


Sutherland, N.C., McDonald J.R. 1989, "Data Models and Spatial Information Systems", Proceedings Inaugural Colloquium of the Spatial Information Research Centre, 67-76, University of Otago, Dunedin


Yourdon, E. 1990, "Auld Lang Syne, Is it time for you to ring out the old and ring in the new?", BYTE Magazine, October , 257-263
Minerva Access is the Institutional Repository of The University of Melbourne

Author/s: Hesse, Walter; Hazelton, William; Williamson, Ian P.

Title: Object-oriented concepts for software development and conceptual modelling in GIS and surveying

Date: 1993


Publication Status: Published

Persistent Link: http://hdl.handle.net/11343/34018

File Description: Object-oriented concepts for software development and conceptual modelling in GIS and surveying

Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.