Object calculus and the object-oriented analysis and design of an error-sensitive GIS

MATT DUCKHAM matt@cs.keele.ac.uk
Department of Computer Science, University of Keele

Keywords: Object-oriented analysis, object-oriented design, object calculus, formal methods, data quality

Correspondence address

Matt Duckham
Department of Computer Science
University of Keele
Staffordshire
ST7 8AA, UK
Email: matt@cs.keele.ac.uk
Tel: +44 1782 584270
Fax: +44 1782 713082

Acknowledgements

This research was funded as part of a Natural Environmental Research Council CASE studentship at the University of Glasgow, sponsored by Survey and Development Services (SDS). Supervision for the studentship was from Dr Jane Drummond and Dr David Forrest of Glasgow University Geography and Topographic Science Department and from John McCreadie at SDS. Gothic software was kindly supplied under a development license by LaserScan, UK. The help, interest and advice of Dr Tom Melham of Glasgow University Computing Science Department is gratefully acknowledged. Finally, the extensive and constructive comments of the three anonymous reviewers were extremely influential in shaping this paper.

Object calculus and the object-oriented analysis and design of an error-sensitive GIS

Abstract. The use of object-oriented analysis and design (OOAD) in GIS research, development and application is now well established. However, the tendency towards informality in OOAD techniques means many of the potential benefits of object-orientation (OO) are being discounted. Recent advances in the formal representation of OO systems may help realise these benefits with respect to some of the key contemporary issues in GIS. This paper examines the application of one particular OO formalism, the ς (sigma) calculus of Abadi and Cardelli, to a long-standing unresolved research problem in GIS: the development of an error-sensitive GIS. The work indicates that significant enhancements in the exploration, verification and understanding of OO systems can be achieved through the use of ς-calculus in support of conventional OOAD techniques.

1. Introduction

Object-orientation (OO) is increasingly the dominant information systems (IS) development paradigm. Notwithstanding this success, the term object-orientation remains highly nebulous and there exists no single, concise definition of the term [61]. The fundamental features of OO are often taken to be classification, encapsulation and inheritance. Classification deals with the complexity of a problem from the top down and focuses on the essential features of an object in the problem domain which distinguish it from other objects. Complementary to classification, encapsulation approaches complexity from the opposite direction and aims to conceal the detailed mechanisms and features of an object. Finally, inheritance allows classified, encapsulated objects to be structured in a hierarchy. The hierarchy allows blueprint features to be described just once, and inheriting classes can then incrementally specialise these blueprint features.

Unfortunately, while the core concepts of classification, encapsulation and inheritance are likely be mentioned in any discussion of OO, the term evades an exhaustive definition. A range of related concepts, such as polymorphism, persistence and identity, may be arrayed alongside the core concepts. Further, the individual concepts themselves may be subject to a variety of interpretations depending on the context. This paper aims to look at some of the reasons for, and practical consequences of this ambiguity. Using the example development of an OO geographic information system (GIS), recent advances in object theory are applied to the OO development process to resolve this indeterminacy.

1.1. OOA, OOD and OOP

Object-orientation has had a substantial impact upon GIS specifically, as it has on IS generally. Most new GIS make significant, if not exclusive use of OO and OO has been used widely in research for a variety of GIS architectures [8, 24, 32, 34, 44, 50, 52, 54].

, , 3–27 ()

Candidate reasons for the enormous success of OO are many fold. For the programmer, object-oriented programming (OOP) offers highly efficient code organisation that is able to boost productivity and promote code reuse beyond anything achievable with even the best possible modular programming [42]. For the designer, object-oriented design (OOD) translates into efficiency of concept, allowing larger and more complex software projects to be implemented and maintained. For the analyst, object-oriented analysis (OOA) promises an implementation independent mechanism for resolving the complexity of the problem domain into abstracted, comprehensible systems which are arguably closer to our intuitive understanding of the world around us [49]. However, when taken together, it is the use of the OO paradigm as a single, consistent heuristic throughout the entire process of OO analysis, design and programming, which is arguably central to the success of the OO revolution [33]. Object-orientation promises, for the first time, a one-stop solution for IS development: the same paradigm can be used from problem definition right through to a working software solution.

1.2. Shortfalls of OO development

Despite this allure, the OO development process does suffer some shortfalls. The enormous success of OO is correlated with a proliferation of OO technology, but this proliferation has not always been complemented by a growth in OO theory. The surfeit of object-oriented analysis, design and programming techniques which exist are, therefore, necessarily highly subjective. Experience, personal preferences and choice of object-oriented analysis and design (OOAD) technique and programming language will all play an important role in the shape of the software engineered.

A clear line is often drawn between the purpose of analysis, which aims to describe *what* a system is supposed to do, and design, which aims to describe *how* a system performs this function [53]. In principle, the results of an OOA are implementation independent whilst OOD is dependent on a target implementation. However, the distinction is much less clearly defined in the practical application of OOA and OOD. Most authors acknowledge the existence of a "continuum of representation" [19] where the implementation independent OOA process blurs with the implementation dependent OOD process [21, 46, 48].

The existence of theory-deficient software engineering techniques may not be of concern in many applications. However, even within the highly results led commercial sector, the use of more formal methods to guide and document OO software development is gathering acceptance [11]. Rather than focus entirely on the production of software, academic research very often aims to make more general statements about the problem domain being studied using OO. It should come as no surprise, then, that a variety of attempts have been made to provide formal methods to support OO technology and to formulate a comprehensive theory of objects.

1.3. A theory of objects

The development of the ς (sigma) calculus by Martín Abadi and Luca Cardelli, resulting in the publication of *A theory of objects* in 1996 [4], represents the one of the most comprehensive attempts to date to provide a formal algebraic description of object systems. A number of other attempts have been made; λ (lambda) calculus, the foundation of functional programming, has been used with limited success, for example, in [15, 25]. However, since λ-calculus uses the function as a primitive construct, object calculi based on λ-calculus quickly tend towards high levels of complexity [4]. Generally, the reuse in an OO context of existing formalisms, such as λ-calculus, first order logic [24] and co-algebras [38], does result in efficiency of concept, but can quickly become prohibitively complex. The ς-calculus of Abadi and Cardelli makes use of objects as a primitive construct and is arguably able to express more fully the features of object systems as a consequence.

The existence of a fundamental theory of objects presents the possibility of addressing some of the shortfalls of OO technology. Object-oriented theory has, in effect, caught up with object-oriented technology. The application of ς-calculus within the OOAD process is currently largely unexplored. This paper details the use of ς-calculus to support the development of a GIS able to manage uncertain spatial information, and uses this example to illustrate how the ς-calculus may be of wider use in OO GIS development.

2. Towards an OO error-sensitive GIS

Despite two decades of research, progress toward the production of a GIS able to handle uncertain spatial information, often termed an *error-sensitive GIS* [55], has been slow. A number of authors have noted that commercial GIS have little or no data quality management capabilities [7, 12]. While increasing emphasis on data warehousing for geographic information (GI) is beginning to result in greater availability of meta-data management tools for GIS [29], generic error-sensitive GIS functionality remains elusive. This section provides a review of existing approaches to the development of error-sensitive GIS, and introduces an OO conceptual model of error in GI that aims to overcome some of the difficulties encountered in previous work.

2.1. Uncertainty management in GIS

The usual mechanism for reporting the uncertainty associated with GI is through the medium of data quality. The use of data quality within GIS hopes to allow data users to assess how suitable a given data set is for a particular purpose, termed *fitness for use* [17]. A variety of data quality standards have grown up under this banner. The United States' National Committee for Digital Cartographic Data Standards (NCDCDS) has been hugely influential in the development of data quality standards. Most subsequent standards have, at their core, made use of the five data quality elements defined by the NCDCDS: lineage, attribute accuracy,

positional accuracy, logical consistency and completeness [47]. A survey of the 22 national data standards included in [45], for example, reveals that most have inherited their approach to data quality from the NCDCDS standard. Despite this broad agreement, the exact composition of data quality standards and definition of the individual elements within different standards can exhibit wide variation. It has been noted that no single data quality standard can claim to be either exhaustive representations of data quality nor exclusive of other standards and approaches [22]. Standards are subject to continual revision and improvement. Though scarce, many previous attempts to implement uncertainty management within GIS have tended to be data quality standard led (eg [32, 57]). In the light of the volatile nature of data quality standards, the standards-led approach, where standards are effectively "hard-wired" into software design, is open to criticism.

Where standards have tended to dominate research into data quality concepts, research into the architectural aspects of error-sensitive in GIS has tended to be dominated by the "quality sub-system" approach, typified by [41, 51, 59]. The approach is important as it aims to maximise software reuse through the addition of a separate quality sub-system, offering error handling routines and modules, to existing GIS. However, the development of a quality sub-system does have drawbacks. First uncertainty is usually thought of as a "fundamental" feature of GI [18, 28]. Consequently, the artificial dichotomy between error and GI introduced by a quality sub-system can be correlated with additional conceptual and computational complexity in the system. Second, separate quality sub-systems tend to result in global or generalised quality information being applied to blocks of GI, for example on a per-layer basis [41]. The inadequacies of such global, generalised data quality information are well documented, for example with reference to DEMs [26]. Consequently, a more general approach to uncertainty able to support the integration of data quality and GI without being tied to any particular data quality standard is sought here.

2.2. Conceptual OO model of data quality

In attempting to redefine data quality without direct reference to existing data quality standards, help does exist in the literature. The conventional conceptual model of GIS is summarised in Figure 1 (see [20, 39, 43, 58, 60]). The assumption made in Figure 1 is that the infinite complexities of the real world can only be usefully handled following the abstraction of the real world to an idealised abstract specification. An actual data set or database constitutes a representation of this abstract specification. However, in contrast to geographic objects, there is no meaningful concept of data quality in the real world. Data quality is not usefully modelled by the processes of abstraction and representation, rather it is only as a side effect of deficiencies in these processes that data quality arises at all [20, 58].

[Figure 1]

In attempting to maximise software reuse without compromising the underlying data model, OO represents a mechanism able to avoid the dichotomous, globalised data quality models outlined above. Since OO uses the concepts of object and class

as primitive notions, OO should be able to address uncertainty at every level of the database, rather than just for blocks of GI. It is important to note that the use of objects in an object-oriented sense in no way implies the use of objects in the sense of the object versus fields debate in GIS (see [27]). OOA offers a technique for modelling concepts, and can equally well be used to model continuous spatial fields as discrete spatial objects. At some level discretisation is a necessary component of any computational system: even continuous fields or functions can only be represented in a computer following some discretisation, for example into the discrete parameters of a polynomial function. However, conceptual discretisation cannot generally be inferred from computational discretisation.

The combination of the conceptual model in Figure 1 and OO software development techniques seem well suited to developing error-sensitive GIS able to address the problems evident in existing approaches to data quality. Further, the use of OOAD promises a direct route from the conceptual model to OO software which embodies the properties of this model. Details of the OOA process can be found in a wealth of literature on the subject (eg [10, 19, 53]). Software based on this analysis should be flexible enough to support not simply existing data quality standards, but potentially any quality standard or elements which might reasonably be proposed and at any level of the database. Severing the close link between software and standards, posited in §2.1, should result in increased flexibility and greater ability to address the real issue underlying most standards, that of fitness for use. Such software should assist in restoring the balance of emphasis toward the awareness, understanding and use of data quality information rather than enforcing compliance with any one particular standard upon which the software happens to be based.

2.3. OOA of data quality

An OOA of the conceptual model in Figure 1 was conducted as the first stage in the development of an error-sensitive GIS. The analysis process highlighted a general distinction that can be made between what is termed here *abstractive quality* and *representative quality*. Abstractive quality results from the process of abstraction in Figure 1. From an OO perspective, abstractive quality operates solely at the class level, because it refers to the inconsistent and incomplete classification of objects. In contrast, representative quality results from the process of representation in Figure 1. From an OO perspective, representative quality operates solely at the object level, because it results from the gap between particular instances of classes when compared with the idealised abstract specification. For example, topological consistency is used in vector GIS to enforce topological relations between different classes of geographic objects. In a database of road and river features, all bridges may be topologically consistent, say, if they fall on the intersection of two roads or of a river and a road. This type of topological consistency can be thought of as an abstractive quality element, as its behaviour will be identical for every object of a particular class, in this case bridges. In contrast, the accuracy of location of a bridge in this hypothetical database may be unique to that bridge. Different

survey methods, data collection and data provenance will mean that positional accuracy may vary for different objects in the same class. It follows that this type of positional accuracy is an example of a representative quality element. In fact, spatial data quality assessment often demands more complex, aggregated quality elements than those in the simplified example above. The strength of using the concepts of abstractive and representative quality is that they can be instructive in decomposing more complex quality elements into their simpler constituents.

Such general results are helpful starting points in the construction of an error-sensitive GIS. The following section looks in more detail at the development of an OO error-sensitive GIS. In particular, the role of ς-calculus in extending and supporting the conventional OO development process is introduced and explored.

3. Object calculus

The discussion above indicates some of the potential advantages of conventional OO development techniques for error-sensitive GIS. On the basis of the framework of ideas sketched out in §2 it would be possible to develop an OO GIS with error handling capabilities using conventional OO development techniques. However, as suggested in §1.2, the partially subjective nature of conventional OO development techniques provides companies and researchers with only limited control over the movement from concepts to software. This section introduces the use of ς-calculus as a supplement to conventional OO development techniques. Focusing on three areas of error-sensitive GIS development in turn, the following sections explore the use of ς-calculus with respect to conflict resolution, system properties and expressive and notational power. Finally, §3.4 proposes a general typology of the uses of ς-calculus.

While enough of the ς-calculus is presented below to support the examples used, a full exploration of the formalism would be impossible here and the reader is directed to the literature for a more substantial treatment of the ς-calculus [2, 3, 4, 5]. As far a possible, the notation used in the following examples is consistent with this literature to facilitate the reference.

3.1. Schema development

An important feature of any formal method in software development is the ability to capture the results of an analysis in a precise, unambiguous manner [40]. A class diagram representing a simplified core of results from the conventional OOA process is shown in Figure 2. While often supported with text, class diagrams such as Figure 2 are central to the communication of conventional OOA results [11]. The graphical notation used in Figure 2 is based on object modeling technique (OMT) [53]. A class diagram defines important classes of objects and their behaviours. Additionally, inheritance relations between classes are given, showing where one class is a specialisation (sub-class) or generalisation (super-class) of another. In Figure 2 the classes **uncertain objects** and **abstraction** are used to model the general error-sensitive properties induced by representative and abstractive quality respectively. The **get rep** method enables **uncertain objects** to be interrogated about their rep-

resentative quality. In a similar manner, the class **abstraction** provides a template for abstractive quality elements. In Figure 2, the **abstraction** class contains an **is consistent** method which allows instances of the class to be interrogated about their logical consistency. The ellipsis in **abstraction** highlights that **is consistent** is just one of many possible abstractive quality methods. The class **geographic objects** is meant to represent the super-class of a conventional OO schema for GI, and itself inherits from **uncertain objects** and **abstraction**. In this way, OO can be used to transmit the general error-sensitive behaviours encapsulated within **uncertain objects** and **abstraction** throughout a conventional OO spatial database: an OO approach is still able to maximise software reuse without the need for separate quality sub-systems. A fourth class, **representative quality** both inherits from and is associated with **uncertain objects**. The class **representative quality** is intended as the generic super-class of all representative quality elements.

[Figure 2]

3.1.1. Untyped ς-calculus Even for results in this simplified form the ς-calculus can be useful for supporting the OOAD process. A formal representation of the classes in Figure 2 requires the reformulation of the class diagram as ς-calculus terms. Objects within the ς-calculus are represented as collections of methods l_i each with bodies b_i. The symbol ς is used to bind the postfixed 'self' parameter (conventionally s or z) with occurrences of that parameter in the body of the method. Each object is enclosed in square brackets and associated with a label using the symbol $\triangleq$ as in Equation 1.

$$o \triangleq [l_i = \varsigma(s)b_i \ ^{i\in 1..n}] \tag{1}$$

Informally, Equation 1 defines a new object o which is a collection of n methods, each with distinct labels l_i and bodies b_i in which references to the object itself may occur, using the bound variable s. A method l of an object o can be invoked using the dot operator, written $o.l$. A class C can be represented as an object with a *new* method, as in Equation 2.

$$\begin{aligned} C \triangleq [new = \varsigma(z)[l_i = \varsigma(s)z.l_i(s) \ ^{i\in 1..n}], \\ l_i = \lambda(s)b_i \ ^{i\in 1..n}] \end{aligned} \tag{2}$$

Invocation of the *new* method on C produces a new object where each of the template or pre-methods, $l_i = \lambda(s)b_i \ ^{i\in 1..n}$, in the class C are bound to methods in the object being created. The term $l = \lambda(s)b$ is in fact shorthand for $l = \varsigma(z)\lambda(s)b$ where z is unused. The use of λ (lambda) in the pre-methods betrays the development of ς-calculus from λ-calculus and the λ performs an analogous binding function to ς. In this way it is possible to obtain a direct mapping from the classes in Figure 2 to an ς-calculus object system. The ς-calculus terms for *Unc* (Definition 1) and *Abs* (Definition 2) are formal representations of the classes **uncertain objects** and **abstraction** respectively.

Definition 1

$$Unc \triangleq [new = \varsigma(z)[get\_rep = \varsigma(s)z.unc_1(s)], \\ unc_1 = \lambda(s)b_1]$$

Definition 2

$$Abs \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.abs_1(s)], \\ abs_1 = \lambda(s)b_2]$$

Inheritance is needed to allow the object system to be extended to include a *Geo* term, corresponding to the class **geographic objects** in Figure 2. Inheritance can be represented by re-using pre-methods, as illustrated by the ς-calculus term for *Geo* in Definition 3. A similar term for the **representative quality** class *Rep* can be constructed as in Definition 4.

Definition 3

$$Geo \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.geo_1(s), \\ get\_rep = \varsigma(s)z.geo_2(s)], \\ geo_1 = Abs.abs_1, \\ geo_2 = Unc.unc_1]$$

Definition 4

$$Rep \triangleq [new = \varsigma(z)[get\_rep = \varsigma(s)z.rep_1(s)], \\ rep_1 = Unc.unc_1]$$

3.1.2. Conflict resolution During the OOAD process conflicts inevitably occur. For example, the error-sensitive GIS described here was implemented within two separate environments, Java and Laser-Scan Gothic OO GIS. While both Java and Gothic are object-oriented environments, they support very different object semantics. In particular, Gothic allows the inheritance of one sub-class from more than one super-class, termed *multiple inheritance*, while Java does not. Multiple inheritance is often very useful during OOA and generally OOA does not proscribe its use [10, 19]. Since the simplified class diagram in Figure 2 uses multiple inheritance, the Java implementation of the analysis results required a slight modification to the class diagram to remove all multiple inheritance, as shown in Figure 3. However, other modifications are possible: a different schema is obtained if **uncertain objects** is allowed to inherit from **abstraction** rather than **abstraction** inheriting from **uncertain object**. Given that **representative quality** and **geographic objects** represent generic super-classes for potentially very large sub-schema, the effect of such

changes will be felt by a significant number of sub-classes. It is useful, therefore, to have some formal way to exercise control over such conflicts.

Using ς-calculus it is possible to obtain a mapping for the singly inheriting classes shown in Figure 3. The terms for Abs' and Geo' in Definitions 5 and 6 correspond to the the redefined classes **abstraction** and **geographic objects** respectively in Figure 3.

[Figure 3]

Definition 5

$$\begin{aligned} Abs' \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.abs_1(s), \\ get\_rep = \varsigma(s)z.abs_2(s)], \\ abs_1 = \lambda(s)b_2, \\ abs_2 = Unc.unc_1] \end{aligned}$$

Definition 6

$$\begin{aligned} Geo' \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.geo_1(s), \\ get\_rep = \varsigma(s)z.geo_2(s)], \\ geo_1 = Abs'.abs_1, \\ geo_2 = Abs'.abs_2] \end{aligned}$$

Equality between objects (and so classes) is defined in the ς-calculus where two objects have exactly the same methods in any order (written $\leftrightarrow$), although it is worth noting that other more flexible equational theories have also been proposed (eg [31]). By evaluating the terms for abs_1 and abs_2 in Definition 3 and the terms for geo_1 and geo_2 in Definition 6 it is possible to formulate a simple proof that the classes Geo and Geo' are equal ($Geo \leftrightarrow Geo'$), although Abs and Abs' are not.

Conflicts between the analysis and design stages of software development are conventionally resolved in an informal manner during the design and programming of the software. This in itself is not necessarily a problem. Software development is an iterative process and the analysis results are by no means set in stone once the design is underway. However, as additional complexity is added to the system and further incompatibilities between OO environments are taken into account, such informal techniques become less clear-cut and more prone to mistakes. By representing the object schema using the ς-calculus, it is possible to more closely control and manage system complexity and incompatibility.

3.2. Exploration of system properties

The term *meta-quality* refers to quality information about quality [1]. The European Committee on Standardisation's (CEN) draft paper on geographic data quality is part of a more general movement to reporting not only meta-data but meta-quality

information [16]. Even the original NCDCDS standard made some allowances for meta-quality in terms of reporting the methods used to derive data quality information [47]. Once the concept of meta-quality is acknowledged as important, however, there is no particular reason to impose these restrict meta-quality to just one level of self-reference. Preferable to the arbitrary prescription of meta-quality elements and structure would be an error-sensitive GIS design able to support recurse levels of meta-quality assignment.

The ς-calculus can be used to explore the recursive properties of the system in Figure 2. The class **representative quality** itself inherits from **uncertain objects**, consequently inheriting the association with **representative quality** objects. It follows that meta-quality information can be assigned recursively if desired; any representative quality object can itself have representative quality information associated with it. Recursion within object schema has been recognised not only as important tool in the development of OO systems, but also as a potential source of unexpected and undesirable object behaviour particularly when used in conjunction with multiple inheritance [9]. By using the ς-calculus to represent the objects involved, developers can explore the properties of a system and communicate those properties in a formal way to avoid the pitfalls presented by the use of recursion.

3.2.1. Using typing rules within ς-calculus The previous example made use of untyped ς-calculus. However, the introduction of types as a method for categorising object terms can increase the expressive power of the ς-calculus. In the ς-calculus, type annotations are used to convey information about the type of methods and objects. For example,Equation 3 states that type A is composed of methods l_i each of type B_i.

$$A ::= [l_i : B_i^{i \in 1..n}] \tag{3}$$

It is possible to give a type for an object of class *Unc* as in Definition 7. This term defines an uncertain objects type, *UncType*, with one method: a *get_rep* method which returns a representative quality object of type *RepType*.

Definition 7

$$UncType ::= [get\_rep : RepType]$$

It would be desirable to be able to write $RepType ::= [get\_rep : RepType, ...]$ as a similar definition for representative quality objects. Unfortunately, this definition of *RepType* contains a circular reference to *RepType* itself. To deal with recursion in the ς-calculus it is necessary to define a recursive type $\mu(X)[get\_rep : X, ...]$ shown in Definition 8, where μ (mu) binds the type variable X to occurrences of X in the body of the type. A second type, *URepType* is obtained by substituting *RepType* for the type variable X, shown in Definition 9. The two types in Definitions 8 and 9 are isomorphic and the constructs *fold* and *unfold* are used to move between these two types. For some object $o : RepType$ we can write $unfold(o) : URepType$

and $fold(RepType, unfold(o))$: $RepType$. Despite this additional complexity, the effect Definitions 8 and 9 is informally analogous to writing $RepType ::= [get\_rep : RepType, ...]$.

Definition 8

$$RepType ::= \mu(X)[get\_rep : X, ...]$$

Definition 9

$$URepType ::= [get\_rep : RepType, ...]$$

Armed with the recursive type $RepType$ and the unfolding of this type $URepType$ it is possible to explore the properties of this system. For an uncertain object o : Unc the typing rules in ς-calculus enable us to deduce the result of accessing the get_rep method on o as below.

$$o : UncType$$
$$o.get\_rep : RepType$$

It is possible to illustrate the property of meta-quality by attempting to access the representative quality not simply of an uncertain object $o : UncType$ but of the resulting representative quality object itself, shown below.

$$o : UncType$$
$$o.get\_rep : RepType$$
$$unfold(o.get\_rep).get\_rep : RepType$$
$$unfold(unfold(o.get\_rep).get\_rep).get\_rep : RepType$$
$$...$$

Such terms can be used as the basis of a simple inductive proof that the object system supports recursive meta-quality. In practice, the ability to use meta-quality beyond a certain point may be of limited use and the extreme flexibility of the system is achieved at the cost of some redundancy. However, this simple example both introduces the use of type systems and illustrates how such type systems can be used to explore the properties of an object system.

3.3. Expressiveness and notation

Object oriented analysis and design have already been defined as the *what* and *how* of OO system development respectively. Issues such as memory requirements and data volumes are usually considered entirely the preserve of the design stage of

development [21]. However, the storage of data quality elements associated with spatial information presents situations where even these relatively clear cut issues can become enmeshed in the continuum of representation.

Even within the NCDCDS data quality standard, a single datum may be associated with up to four different items of quality information (lineage, consistency, completeness and either positional or thematic accuracy). Subsequent standards such as the 1996 CEN draft standard contain as many as 13 different types of data quality elements, most of which can meaningfully refer simultaneously to the same GI. If the existence of meta-quality information is also taken into account, it is clear that any spatial database, if saturated with data quality information, might increase in size by an order of magnitude or more. The *cost* of digital data storage is continually tumbling and so the the storage of huge volumes of data quality information may be an option in some situations: in the near future terabyte storage media are likely supersede the megabyte and gigabyte storage media in common usage today. However, intuitively the *value* of data quality information is in some way constrained by the *value* of the information to which it refers: data quality 'adds value' to GI. Arguably, an organisation which stores gigabytes of GI is unlikely to ever want to store terabytes of associated quality information, irrespective of how economical data storage becomes. The effort involved in storing thousands of times more data quality information that GI will always be difficult to justify as long as data quality information is not thousands of times more valuable than GI. In short, the design of an error-sensitive GIS has the question of data storage as a *what* as well as a *how*.

3.3.1. Efficient data storage Given that data volume is an analysis issue in the context of developing an error-sensitive GIS, it is possible to develop strategies to minimise data volume. One such strategy takes advantage of the generally hierarchical nature of aggregated objects in an OO spatial database. A highly simplified example of the aggregation relationships which might exist in a vector OO GIS are shown in Figure 4. Here the representation of a river is actually the aggregation of a number of lines, which in turn are aggregates of a number of points. Because each class of spatial objects in the database inherits from the **geographic objects** class in Figure 2, the model used here allows individual representative quality objects to refer to any one of these spatial objects. A first step in minimising data quality volumes is to allow spatial objects in the database to infer quality from the objects of which they are a part. Assuming, say, Line A in Figure 4 has a quality object associated with it, the point objects which make up Line A (Points A, B and C) should be able to infer their quality from the aggregate line.

[Figure 4]

An example of where such precedence rules already enjoy practical application is in US vector product format (VPF) military standard [56]. The ς-calculus can be used to formulate an algebraic representation of the VPF precedence rules, as illustrated by the untyped term for the class Unc' in Definition 10. The notation $o.x \Leftarrow \varsigma(s)b$ is used to replace the body of the method x on object o with $\varsigma(s)b$. The $\varsigma(s)$ is usually omitted where s is unused in b.

Definition 10

$$
\begin{aligned}
Unc' \triangleq [&new = \varsigma(z)[parent = [get\_rep = [\,]], get\_rep = \varsigma(s)z.unc_1(s),\\
&set\_rep = \varsigma(s)z.unc_2(s), add\_parent = \varsigma(s)z.unc_3(s)],\\
&unc_1 = \lambda(s)s.parent.get\_rep,\\
&unc_2 = \lambda(s)\lambda(q)s.get\_rep \Leftarrow q,\\
&unc_3 = \lambda(s)\lambda(p)s.parent \Leftarrow p]
\end{aligned}
$$

Given the definition of Unc' it is possible for objects to infer quality information in the manner suggested by VPF. In Equation 4 reduction of ς-calculus terms is used to show that a child Unc' object is able to infer representative quality from its parent object. The reduction of invoked methods in ς-calculus is indicated using the symbol $\rightarrowtail$, for example $[l = [\,]].l \rightarrowtail [\,]$. A detailed exposition of reduction in ς-calculus terms is beyond the scope of this paper, but the informal semantics of reduction should be fairly intuitive to anyone familiar with OOP.

$$
\begin{aligned}
&rep \triangleq Rep.new\\
&parent \triangleq (Unc'.new).set\_rep(rep)\\
&child \triangleq (Unc'.new).add\_parent(parent)\\
&child.get\_rep \rightarrowtail rep
\end{aligned}
\tag{4}
$$

By using the revised definition of Unc' above in place of the definition of Unc in Definition 1, VPF-like precedence rules can be transmitted to all objects throughout the database. In fact, VPF embodies one example of a wide variety of possible precedence rules. For example, more sophisticated precedence rules are needed where aggregation in geographic objects is not strictly hierarchical, and where one object form part of more than on aggregate object. Similarly, quality inference in Definition 10 goes in only one direction: components infer from aggregates. Equally, it is possible to compose precedence rules where quality in an aggregate object is inferred through some (error propagation) function of its component objects. In practice, the details of such systems quickly become difficult to explain using the traditional OO analysis tools of diagrams and text alone. The provision of a mechanism that allows the detailed workings of a proposed system to be explored algorithmically is therefore of great benefit to the error-sensitive GIS analyst. Indeed, using the ς-calculus proved very useful in this research and led directly to new efficient quality storage algorithms being developed and implemented in the software resulting from this work. The exploration of some of these possibilities is one of the themes of §4. First, §3.4 attempts to solidify some of the ideas introduced in this section.

3.4. Uses of ς-calculus

The example application of ς-calculus in this section illustrates a cross-section of difficulties that may be encountered by the system developer during the transition

from OO analysis to design. Whilst the literature acknowledges the indeterminacy of the boundary between OOA and OOD, underlying much of the OO development process is the assumption that the movement of information and distinction between OOA and OOD is reasonably smooth. This situation is illustrated in Figure 5A, after [46]. Here, information about the problem domain is captured initially with OOA and further information is subsequently captured with OOD. Crucially, the OOD process encompasses all of the OOA results and the boundary between OOA and OOD is distinct. This model of the development process is, however, incomplete. Taking each of the three examples above in turn, a characterisation of the use of object calculus within the OOAD process is proposed based on the deficiencies in the model of the OOAD process shown in Figure 5A.

[Figure 5]

The first of the three example uses of the ς-calculus deals with the representation of multiple inheritance in both OOA and OOD. Information captured about the problem domain in the form of a multiply inheriting object, could not be used in the design without reformulating the multiple inheritance as single inheritance. The general situation where, despite information being captured by the OOA, the design cannot incorporate those features, is shown in Figure 5B. Information can leak out of the development process at the boundary between analysis and design. The ς-calculus can be used to resolve any conflicts resulting from this information leakage, providing a route to reconcile the analysis with the design. Inheritance strategies are only one possible area where such information leakage can occur. As already noted the term object-orientation is nebulous and as a consequence some mismatch between the concepts used in OOA and in OO programming environments is to be expected.

The second example looks at the need for a capacity to explore the properties of the results of an OOA. Within the commercial world, it may be enough to verify that the OO software produced fulfills all the client's requirements, to be satisfied that the development process used to produce that software has been a success. However, it may be unreasonable to infer the properties of an OOA from a software product produced from that analysis. In Figure 5C, despite the OOA being captured in its entirety by more than one design it is not possible to make any assumptions about the general properties of the analysis from those designs. Different designs will extend the analysis results in different ways. Simply because one design and implementation possesses certain properties does not imply that another necessarily will. By using the ς-calculus to formalise and explore the analysis, it is possible to make statements about the core properties of the analysis which, in turn, should be fundamental to any design based on that analysis.

Finally, the ς-calculus was used to provide an expressive, implementation independent platform to communicate specific OOA results. The detailed working of an object system is usually excluded from the OOA process and is often regarded as the *how* of system design rather than the *what* of problem definition. This does encourage the analyst to focus on general rather than implementation issues. However, equally it is the lack of an implementation independent language able to communicate the detailed working which has led to the proscription of detailed

working within OOA. The example of the volume of data quality required to describe the uncertainty associated with spatial information shows that, in some cases, the detailed working of an object system can become a central part of the problem definition. The general situation is illustrated in Figure 5D, where there is no distinct boundary between OOA and OOD. Consequently, no part of the development process is guaranteed to be exclusively the preserve of analysis or of design.

4. Application of ς-calculus

The previous section introduced the ς-calculus and indicated a number of points in the conventional OO development process that can benefit from the support of formal algebraic object calculus. As already mentioned, the combination of conventional and formal approaches was successfully used to develop a prototype implementation within Java and a full implementation within LaserScan Gothic OO GIS. The full implementation exhibited the desired high levels of flexibility, with ability to support recursive meta-quality, efficient quality storage, and was successfully integrated with a range of different data quality standards, namely Spatial Data Transfer Standard (SDTS), Spatial Archive and Interchange Format (SAIF) and CEN/TC287. However, as indicated in §3.4 simply describing the resulting implementation may not provide a clear picture of the efficacy of underlying concepts and techniques used in the analysis and design process. Consequently, rather than offer a detailed, implementation dependent account of the LaserScan Gothic error-sensitive GIS, this section uses the ς-calculus to construct a very simple implementation independent application that illustrates the potential flexibility and utility of the underlying concepts used in the error-sensitive GIS. The object system explored in this section represents the core formal specification of an error-sensitive GIS and is used to provide a 'walk-through' of the different stages of error-sensitive GIS use. The importance of ς-calculus in this process is that it offers a powerful, implementation independent mechanism for algorithmic manipulation of objects and classes which are avatars of those that would exist in a real database application.

4.1. Preliminary assumptions

For notational convenience, the object system discussed below assumes the existence of a number of objects not formally defined here, namely Boolean *true* and *false* objects, real numbers $x \in \mathbb{R}$ and ordered alphanumeric string objects. To highlight the fact that Booleans, reals and strings are still objects, they are written in enclosing square brackets, eg [*true*], [1.0] and ["string"]. A number of standard operators are assumed for these objects. An equivalence binary operator ($\equiv$) is assumed for Booleans, reals and strings (eg ["abc"] $\equiv$ ["abc"] reduces to $[false]$); arithmetic operators are assumed for real numbers (eg $[1.0] + [2.0]$ reduces to [3.0]); a simple *if-then-else* construct is assumed for Booleans (eg if [2.0] $\equiv$ [3.0] then ["Equivalent"] else ["Not equivalent"] reduces to ["Not equivalent"]). While these primitive objects and constructs are not part of the ς-calculus, it is worth noting that pointers on how to build such objects and constructs using pure

ς-calculus can be found in the literature, eg [4]. For simplicity the example uses untyped ς-calculus terms, although the translation of these terms into a typed universe is also possible.

4.2. Database design

Database design is the first step in using both the software and ς-calculus implementations of the error-sensitive GIS. The error-sensitive GIS provides a core of classes necessary to support error-sensitive behaviour, but both the geographic object schema and the quality object schema remain application dependent. A highly simplified schema is used for this example, illustrated by the class diagram in Figure 6, based on the core error-sensitive schema in Figure 2. In addition to the core error-sensitive classes, the class diagram in Figure 6 defines two geographic primitive object classes, **node** and **edge**, and two representative quality classes, **positional accuracy** and **accuracy test**. In conventional OO GIS a selection of geographic primitives usually already exist and such primitives would usually be reused by integrating them with the error-sensitive schema through inheritance rather than being redefined. Reuse of existing geographic primitives was achieved in this way in the LaserScan Gothic implementation. Additionally, the full LaserScan Gothic software implementation maintains references to a range of representative quality information simultaneously. However, for simplicity, this example is concerned only with **uncertain objects** that refer to just one representative quality object at a time.

Having decided upon an error-sensitive database schema, the implementation of the classes in the schema in the ς-calculus error-sensitive GIS involves arriving at ς-calculus terms for each class. In the same way, a real error-sensitive database designer would need to implement schema classes in a platform dependent way, for example using Lull (Laser-Scan User Language) or C in Laser-Scan Gothic. The terms for *Unc*, *Abs*, *Geo* and *Rep* have already been given in Definitions 1, 2, 3 and 4 respectively. A **positional accuracy** class with two methods, *get_rep* inherited from *Rep* and a standard deviation method σ can be constructed using the term *Pos* in Definition 11 below. Similarly, a term *Test* for the representative meta-quality element **accuracy test** can be constructed as in Definition 12.

Definition 11

$$Pos \triangleq [new = \varsigma(z)[get\_rep = \varsigma(s)z.pos_1(s), \sigma = [\,]],\ pos_1 = Rep.rep_1]$$

Definition 12

$$Test \triangleq [new = \varsigma(z)[get\_rep = \varsigma(s)z.test_1(s), value = [\,]],\ test_1 = Rep.rep_1]$$

An outline definition for the geographic primitive class *Node* can be constructed using the ς-calculus term in Definition 13. In Definition 14, the *Edge* term associates

and edge with two nodes, in addition to providing a *length* method which calculates the distance between the two nodes that comprise an edge.

DEFINITION 13

$$\begin{aligned} Node \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.node_1(s),\\ get\_rep = \varsigma(s)z.node_2(s),\\ x = [\,], y = [\,]],\\ node_1 = Abs.abs_1,\\ node_2 = Geo.geo_2] \end{aligned}$$

DEFINITION 14

$$\begin{aligned} Edge \triangleq [new = \varsigma(z)[is\_cons = \varsigma(s)z.edge_1(s),\\ get\_rep = \varsigma(s)z.edge_2(s),\\ length = \varsigma(s)z.edge_3(s),\\ n_1 = Node.new, n_2 = Node.new],\\ edge_1 = Abs.abs_1,\\ edge_2 = Geo.geo_2,\\ edge_3 = \lambda(s)((s.n_1.x - s.n_2.x)^2 + (s.n_1.y - s.n_2.y)^2)^{\frac{1}{2}}] \end{aligned}$$

The definition of these ς-calculus terms mirrors the database design process that precedes the use of any error-sensitive GIS. The error-sensitive GIS provides only enough class definitions to support the core error-sensitive functionality. In the same way as conventional OO GIS allow application specific geographic object schema to be defined, so the error-sensitive GIS allows application specific quality schema to be defined.

4.3. Abstractive quality

Abstractive quality operates solely at the class level. However, abstractive quality behaviours will usually be redefined for each new sub-class of **abstraction**. In order to obtain sophisticated consistency behaviour, the body of the *is_cons* method must be overridden in sub-classes of *Abs*. For example, new *Node* objects initially do not have valid coordinates ($Node.new.x \rightarrowtail [\,]$). The $Node'$ term in Definition 15 updates the body of the *is_cons* method with a new method body that checks its coordinates are not empty objects, [ ].

DEFINITION 15

$$\begin{aligned} Node' \triangleq Node.node_1 \Leftarrow \lambda(s)\underline{\text{if}}\ s.x \equiv [\,]\ \underline{\text{then}}\ [false]\ \underline{\text{else}}\\ (\underline{\text{if}}\ s.y \equiv [\,]\ \underline{\text{then}}\ [false]\ \underline{\text{else}}\ [true]) \end{aligned}$$

It is now possible for a $Node'$ object to report upon its own consistency. In Equation 5, a new node object is created, which by default has empty x and y coordinates. As a result, invocation of the *is_cons* method upon this node reduces to [*false*] and leads to the conclusion that the object is not logically consistent. In contrast, the new node in Equation 6 is first updated with the coordinates of (5.0, 6.0) before the *is_cons* method is invoked, then reducing to [*true*].

$$(Node'.new).is\_cons \rightarrowtail [false] \tag{5}$$

$$(((Node'.new).x \Leftarrow \varsigma(s)[5.0]).y \Leftarrow \varsigma(s)[6.0]).is\_cons \rightarrowtail [true] \tag{6}$$

A slightly more sophisticated use of this approach, illustrated in Definition 16, redefines the *Edge* class in Definition 14 to check whether the edge's two nodes, n_1 and n_2 are logically consistent before reporting on its own consistency.

DEFINITION 16

$$\begin{aligned} Edge' \triangleq Edge.edge_1 \Leftarrow \lambda(s)&\underline{\text{if}}\ s.n_1.is\_cons\ \underline{\text{then}} \\ &(\underline{\text{if}}\ s.n_2.is\_cons\ \underline{\text{then}}\ [true]\ \underline{\text{else}}\ [false]) \\ &\underline{\text{else}}\ [false] \end{aligned}$$

This in turn allows the consistency of new $Edge'$ objects to be checked as in Equation 7 below.

$$\begin{aligned} &node_1 \triangleq (Node'.new.x \Leftarrow [5.0]).y \Leftarrow [6.0] \\ &node_2 \triangleq (Node'.new.x \Leftarrow [10.0]).y \Leftarrow [10.0] \\ &edge \triangleq (Edge'.new.n_1 \Leftarrow node_1).n_2 \Leftarrow node_2 \\ &edge.is\_cons \rightarrowtail [true] \end{aligned} \tag{7}$$

4.4. Representative quality

Classes that inherit from *Unc* inherit the ability to be interrogated about their own representative quality through the *get_rep* method. However, the details of how this representative quality information is organised and managed are highly flexible. The simplest approach is to associate a representative quality object with its referring object directly using the *get_rep* method. In Equation 8 below, a node is associated directly with a positional accuracy object with a standard deviation method σ of 0.5.

$$\begin{aligned} &node = Node'.new.get\_rep \Leftarrow (Pos.new.\sigma \Leftarrow [0.5]) \\ &node.get\_rep.\sigma \rightarrowtail [0.5] \end{aligned} \tag{8}$$

Even such naïve approaches to representative quality can still be very flexible. Meta-quality information can be represented, as in Equation 9 where a node object is annotated both with quality and meta-quality information.

$$\begin{aligned} &pos \triangleq Pos.new.\sigma \Leftarrow [0.5] \\ &test \triangleq Test.new.value \Leftarrow [\text{“deductive estimate”}] \\ &node \triangleq (Node'.new.x \Leftarrow [5.0]).y \Leftarrow [6.0] \\ &node.get\_rep \Leftarrow (pos.get\_rep \Leftarrow test) \end{aligned} \tag{9}$$

Similarly, a single representative quality object may be referred to by a number of different geographic objects, as in Equation 10.

$$\begin{aligned} &pos \triangleq Pos.new.\sigma \Leftarrow [0.5] \\ &node_1 \triangleq (Node'.new.x \Leftarrow [5.0]).y \Leftarrow [6.0] \\ &node_2 \triangleq (Node'.new.x \Leftarrow [10.0]).y \Leftarrow [10.0] \\ &node_1.get\_rep \Leftarrow pos \\ &node_2.get\_rep \Leftarrow pos \\ &node_1.get\_rep \leftrightarrow node_2.get\_rep \end{aligned} \tag{10}$$

By leaving the choice of representative quality schema to the database designer, a much broader concept of data quality can be accommodated by the system. Data quality on, say, fuzzy set membership values for categorical information can be supported in exactly the same manner as for the discussion standard deviation of location above.

4.5. Dynamic data quality

The discussion in §3.3.1 indicated a number of ways in which more efficient storage and management of data quality information can be achieved. Such techniques open the door to a more general treatment of data quality as a dynamic process rather than as static documentation. A key advantage of using the ς-calculus is that a range of more sophisticated behaviours can be easily and algorithmically explored. For example, data quality for geographic phenomena is often likely to be spatially autocorrelated. Such correlation may be implicit in the database: in Equation 8 objects spatially closer to $node_1$ than $node_2$ may have positional accuracies closer to that of pos_1 rather than pos_2. Occasionally, information about autocorrelation in data quality may exist in an explicit form, for example as a continuous error field. As argued in §2.2, both objects and fields may be modelled using object-oriented techniques. A continuous positional accuracy field, say, can be though of as a method $\varsigma(s)\lambda(x)\lambda(y)b_3\{s,x,y\}$ which uses an (*x*,*y*) coordinate to derive a standard deviation for the point location described by the coordinate. The notation

$b_3\{s, x, y\}$ is used to highlight that s, x and y may occur bound b_3. Definition 17 gives a new field-based definition of the positional accuracy class Pos'.

Definition 17

$$\begin{aligned} Pos' \triangleq\ & [new = \varsigma(z)[get\_rep = \varsigma(s)z.pos_1(s), \sigma = \varsigma(s)z.pos_2(s)], \\ & pos_1 = Rep.rep_1, \\ & pos_1 = \lambda(s)\lambda(x)\lambda(y)b_3\{s, x, y\}] \end{aligned}$$

Equation 11 below, indicates how this new class might be used in combination with two node objects. In contrast to Equation 10, despite both nodes $node_1$ and $node_2$ referring to the same positional accuracy field object *pos_field*, application of the *get_rep* method on the two nodes will not necessarily reduce to the same result as positional accuracy is determined dynamically on the basis of each nodes' (x,y) coordinate. Support for field-based information about error is important as the approach is congruent with a significant body of work into error and error propagation by Gerard Heuvelink (see [35, 36, 37]).

$$\begin{aligned} & pos\_field \triangleq Pos'.new \\ & node_1 \triangleq (Node'.new.x \Leftarrow [5.0]).y \Leftarrow [6.0] \\ & node_2 \triangleq (Node'.new.x \Leftarrow [10.0]).y \Leftarrow [10.0] \\ & node_1.get\_rep \Leftarrow \varsigma(s)pos\_field.\sigma(s.x, s.y) \\ & node_2.get\_rep \Leftarrow \varsigma(s)pos\_field.\sigma(s.x, s.y) \end{aligned} \tag{11}$$

4.6. Error propagation

A logical progression for the error-sensitive GIS, then, is to move from support for dynamic data quality to supporting error propagation more generally. A simple example would be the alteration of the *length* method in the *Edge* class to derive both Euclidean length and the standard deviation of that length. The calculation for standard deviation of length can, in this case, be derived from standard variance propagation techniques not discussed here (see [13]).

Definition 18

$$\begin{aligned} Edge'' \triangleq\ & Edge.edge_3 \Leftarrow \\ & [length = \lambda(s)((s.n_1.x - s.n_2.x)^2 + (s.n_1.y - s.n_2.y)^2)^{\frac{1}{2}}, \\ & \sigma = ((s.n_1.get\_rep.\sigma)^2 + (s.n_2.get\_rep.\sigma)^2)^{\frac{1}{2}}] \end{aligned}$$

Given the term for $Edge''$ in Definition 18, it is now possible to write terms for a simple object system capable of both storage and propagation of error, as in Equation 12. Invocation of the *length* method on the *edge* object in the final line

produces a new object $[length = [6.4], \sigma = [0.86]]$ detailing the length of the edge to be 6.4 with a standard deviation σ of 0.86 (variance of 0.74).

$$\begin{aligned}
&pos_1 \triangleq Pos.new.\sigma \Leftarrow [0.5] \\
&pos_2 \triangleq Pos.new.\sigma \Leftarrow [0.7] \\
&node_1 \triangleq (Node'.new.x \Leftarrow [5.0]).y \Leftarrow [6.0] \\
&node_2 \triangleq (Node'.new.x \Leftarrow [10.0]).y \Leftarrow [10.0] \\
&node_1.get_rep \Leftarrow pos_1 \\
&node_2.get_rep \Leftarrow pos_2 \\
&edge \triangleq (Edge'.new.n_1 \Leftarrow node_1).n_2 \Leftarrow node_2 \\
&edge.length \rightarrowtail [length = [6.4], \sigma = [0.86]]
\end{aligned} \tag{12}$$

Such dynamic error propagation functionality offers considerable further potential. The concept of a measurement-based GIS (M-BGIS), where original survey measurements are retained to allow subsequent least squares adjustment of derived features in the light of resurveys or additional information, has been propounded by a number of authors [14, 30]. Potentially, M-BGIS could be developed along identical lines to the error-sensitive GIS described here.

5. Conclusions and future work

The application of OO techniques to GIS development has already proved very successful, but has considerable further benefits to offer. In order to realise the full potential of OO GIS, the introduction of increased formality into the OO GIS development process is a logical step. The work on formal object systems discussed in this paper has practical implications for two important research areas: first, for error-sensitive GIS development and second for formal OO systems development more generally.

5.1. OO Error-sensitive GIS

This paper has shown that through the medium of ς-calculus relatively sophisticated error-sensitive behaviour can be achieved based on a relatively simple OO conceptual model. The approach seems able to address the difficulties that have dogged previous attempts to implement error-sensitive GIS, namely the reliance of data quality standards and on quality sub-system architectures. The emphasis on an algorithmic formalisation of error-sensitive GIS allows a corresponding emphasis on data quality as a dynamic process. This in turn facilitates the integration of features, such as error propagation, with mainstream GIS functionality, providing a seamless blend of geographic information, quality information and error propagation. Considerable potential exists for further research in this area. In addition to the variance propagation illustrated here, other analytical, geostatistical or stochastic error propagation techniques should also be straightforward to

integrate. One logical way to approach this would be to provide generic superclasses, such as **uncertain objects**, with generic error propagation techniques, such as Monte-Carlo simulation. Using inheritance, these generic techniques could be overridden by more efficient focussed analytical techniques, such as those described in [36], by sub-classes where appropriate. As suggested in §4.6, M-BGIS represent a logical conclusion of this progression, although much more work remains before the system described here could be called an M-BGIS.

It is important to note that the system presented in this paper can only hope to provide a flexible framework for the handling of uncertain spatial information. The actual application of the information by human users to real problems is likely to require a variety of higher level techniques and software (eg [6, 23]).

5.2. OO system development

The ς-calculus of Abadi and Cardelli [4] is more than capable of fulfilling the requirements for a rigorous tool able to support the conventional OO development process. The idea of a language fundamental to all object systems, able to express the entire scope of OO concepts, promises the re-unification and solidification of often fragmented and nebulous OO concepts. By allowing analysts to explore the properties of an object system before design and implementation begin should provide for greater code and concept reuse and for earlier detection of mistakes. Specifically, this work highlights a three distinct situations where formality may be useful in the progression from OOA to OOD: a design may not capture the analysis upon which it is based in its entirety; different designs may implement different features not covered by an analysis; the boundary between analysis and design is not always crisp.

Undoubtedly, the foremost drawback with the ς-calculus as presented here is its complexity especially when compared with the relatively intuitive traditional OOAD techniques. It is likely that for many applications, the simplicity of conventional development techniques outweighs the disadvantages of informality. However, the ς-calculus is arguably no more complex than the OOP languages which many GIS researchers and practitioners use. Further research might productively focus on the derivation of simpler formal development tools from ς-calculus. Abadi and Cardelli [4] present the roots of a programming language based on the ς-calculus, and developments in this direction may lead to an object calculus which is more intuitive. Nevertheless, even in its present form, the ς-calculus has shown itself to be of significant benefit to the process of OOA in the context of engineering an error-sensitive GIS. The informality of OOAD techniques is already being questioned widely. Formality is no guarantee correctness, and conventional OOAD techniques must remain at the core of any OO system development. However, the future of research and development into OO GIS would be well served by integrating the rigorous support mechanisms offered by the ς-calculus.

References

1. H.J.G.L. Aalders. Quality metrics for GIS. In M.J. Kraak and M. Molenaar, editors, *Advances in GIS research II; Proceedings Seventh International Symposium on Spatial Data Handling*, volume 1, pages 5B1–10, 1996.
2. M. Abadi and L. Cardelli. An imperative object calculus. *Theory and Practice of Object Systems*, 1(3):151–166, 1995.
3. M. Abadi and L. Cardelli. A theory of primitive objects: Second-order systems. *Science of Computer Programming*, 25(2-3):81–116, 1995.
4. M. Abadi and L. Cardelli. *A theory of objects*. Springer-Verlag: New York, 1996.
5. M. Abadi and L. Cardelli. A theory of primitive objects: Untyped and first-order systems. *Information and Computation*, 125(2):78–102, 1996.
6. A. Agumya and G.J. Hunter. Determining fitness for use of geographic information. *ITC Journal*, (2):109–113, 1997.
7. R.J. Aspinall. Measurement of area in GIS: A rapid method for assessing the accuracy of area measurement. In *Proceedings of the GIS Research UK 1996 conference*, pages 135–142, 1996.
8. L. Becker, A. Voigtmann, and K.H. Hinrichs. Developing applications with the object-oriented GIS-kernel Goodac. In M.J. Kraak and M. Molenaar, editors, *Advances in GIS research II; Proceedings Seventh International Symposium on Spatial Data Handling*, volume 1, pages 5A1–18, 1996.
9. G. Blaschek and J.H. Frölich. Recursion in object-oriented programming. *Journal of Object-Oriented Programming*, 11(7):29–35, 1998.
10. G. Booch. *Object oriented analysis and design with applications*. Benjamin–Cummings: California, second edition, 1994.
11. J. Bowen. *Formal specification and documentation using Z: A case study approach*. International Thomson Computer Press: London, 1996.
12. C. Brunsdon and S. Openshaw. Simulating the effects of error in GIS. In P.M. Mather, editor, *Geographical information handling: Research and applications*. Wiley: New York, 1993.
13. P.A. Burrough and R.A. McDonnell. *Principles of geographical information systems*. Clarendon: Oxford, second edition, 1998.
14. G. Campbell, L. Carker, and P. Egesborg. A GIS-based multipurpose digital cadastre for Canada lands. In *FIG Congress XX*, 1994.
15. L. Cardelli. A semantics of multiple inheritance. In G. Goos and J. Hartmanis, editors, *Semantics of data types*, number 173 in Lecture notes in computer science, pages 51–67. Springer-Verlag: New York, 1984.
16. CEN/TC287. Draft european standard; geographic information — quality. Technical Report prEN 287008, European Committee for Standardisation, 1996.
17. N.R. Chrisman. The role of quality information in the long term functioning of a geographic information system. In *Proceedings Auto-Carto 6*, volume 1, pages 303–312, 1983.
18. N.R. Chrisman. The error component in spatial data. In D.J. Maguire, M.F. Goodchild, and D.W. Rhind, editors, *Geographical information systems*, volume 1, chapter 12, pages 165–174. Longman: Essex, 1991.
19. P. Coad and E. Yourdon. *Object-oriented analysis*. Yourdon Press: New Jersey, 1991.
20. B. David, M. van den Herrewegen, and F. Salgé. Conceptual models for geometry and quality of geographic information. In P.A. Burrough and A.U. Frank, editors, *Geographic objects with indeterminate boundaries*, GIS Data 2. Taylor and Francis: London, 1996.
21. D. de Champeaux and P. Faure. A comparative study of object-oriented analysis methods. *Journal of Object-Oriented Programming*, 5(1):21–33, 1992.
22. M. Duckham and J. Drummond. Implementing and object-oriented approach to data quality. In B. Gittings, editor, *Integrating Information Infrastructures with GI Technology*, pages 53–64. Taylor and Francis, London, 1999.
23. M. Duckham and J. McCreadie. An intelligent, distributed, error-aware object-oriented GIS. In W. Shi, M.F. Goodchild, and P.F. Fisher, editors, *Proceedings 1st International Symposium on Spatial Data Quality*, pages 596–505, 1999.

24. M.J. Egenhofer and A.U. Frank. Object-oriented modeling: Inheritance and propagation. In *Proceedings Auto-Carto 9*, pages 588–598, 1989.
25. J. Fiadeiro and T. Maibaum. Describing, structuring and implementing objects. In G. Goos and J. Hartmanis, editors, *Foundations of Object-Oriented Languages*, number 489 in Lecture notes in computer science, pages 274–310. Springer-Verlag: New York, 1991.
26. P.F. Fisher. Is GIS hidebound by the legacy of cartography? *Cartographic Journal*, 35(1):5–9, 1998.
27. M. Goodchild. Integrating GIS and spatial data analysis: Problems and possibilities. *International Journal of Geographical Information Systems*, 6(5):407–423, 1992.
28. M.F. Goodchild. Sharing imperfect data. In H.J. Onsrud and G. Rushton, editors, *Sharing geographic information*, pages 413–425. Rutgers: Jersey, 1995.
29. M.F. Goodchild. Directions in GIS. In *Proceedings Third International Conference/Workshop on Integrating GIS and Environmental Modeling*, 1996. http://www.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/main.html.
30. M.F. Goodchild. Measurement-based GIS. In W. Shi, M.F. Goodchild, and P.F. Fisher, editors, *Proceedings of the International Symposium on Spatial Data Quality*, pages 1–9, 1999.
31. A.D. Gordon and G.D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings 23rd Annual ACM Symposium on Principles of Programming Languages*, pages 386–395, 1996.
32. S.C. Guptill. Inclusion of accuracy data in a feature based object oriented data model. In M.F. Goodchild and S. Gopal, editors, *Accuracy of spatial databases*, chapter 8, pages 91–98. Taylor and Francis: London, 1989.
33. W. Haythorn. What is object-oriented design? *Journal of Object-Oriented Programming*, 7(1):67–78, 1994.
34. J.R. Herring. TIGRIS: a data model for an object-oriented geographic information system. *Computers and Geosciences*, 18(4):443–452, 1992.
35. G.B.M. Heuvelink. *Error propagation in quantitative spatial modelling: Applications in Geographical Information Systems*. University of Utrecht, 1993.
36. G.B.M. Heuvelink. *Error propagation in environmental modelling with GIS*. Research monographs in GIS. Taylor and Francis: London, 1998.
37. G.B.M. Heuvelink, P.A. Burrough, and A. Stein. Propagation of errors in spatial modeling in GIS. *International Journal of Geographical Information Systems*, 3(4):303–322, 1989.
38. B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C.B. Jones, C. Lengauer, and H-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer, Massachusetts, 1996.
39. W. Kainz. Logical consistency. In S.C. Guptil and J.L. Morrison, editors, *Elements of spatial data quality*, chapter 6, pages 109–137. Elsevier Science: Oxford, 1995.
40. K. Lano. *Formal object-oriented development*. Springer-Verlag, Berlin, 1995.
41. D.P. Lanter and H. Veregin. A research paradigm for propagating error in layer-based GIS. *Photogrammetric Engineering and Remote Sensing*, 58(6):825–833, 1992.
42. J.A. Lewis, S.M. Henry, D.G. Kafura, and R.S. Schulman. On the relationship between the object-oriented paradigm and software reuse: An empirical investigation. *Journal of Object-Oriented Programming*, 5(4):35–41, 1992.
43. D.J. Maguire and J. Dangermond. The functionality of GIS. In D.J. Maguire, M.F. Goodchild, and D.W. Rhind, editors, *Geographical information systems*, volume 1, chapter 21, pages 319–335. Longman: Essex, 1991.
44. P. Milne, S. Milton, and J.L. Smith. Geographical object-oriented databases: A case study. *International Journal of Geographical Information Systems*, 7(1):39–55, 1993.
45. H. Moellering, editor. *Spatial database transfer standards 2: Characteristics for assessing standards and full descriptions of the national and international standards in the world*. Elsevier Science: Oxford, 1997.
46. D.E. Monarchi and G.I. Puhr. A research typology for object-oriented analysis and design. *Communications of the ACM*, 35(9):35–47, 1992.
47. National Committee for Digital Cartographic Data Standards. The proposed standard for digital catographic data. *American Cartographer*, 15(1):11–142, 1988.

48. J.M. Nerson. Applying object-oriented anlysis and design. *Communications of the ACM*, 35(9):63–74, 1992.
49. C. Partridge. Modelling the real world: Are classes abstractions or objects? *Journal of Object-Oriented Programming*, 7(7):39–45, 1994.
50. B.A. Ralston. Object oriented spatial analysis. In S. Fotheringham and P. Rogerson, editors, *Spatial analysis and GIS*, pages 165–186. Taylor and Francis: London, 1994.
51. B. Ramlal and J.E. Drummond. A GIS uncertainty subsystem. In *Archives ISPRS Congress XVII*, volume 29.B3, pages 356–362, 1992.
52. J. Raper and D. Livingston. Development of a geomorphological spatial model using object oriented design. *International Journal of Geographical Information Systems*, 9(4):359–384, 1995.
53. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice Hall: New Jersey, 1991.
54. A.Y. Tang, T.M. Adams, and E.L. Usery. A spatial data model design for feature-based geographical information systems. *International Journal of Geographical Information Systems*, 10(5):643–659, 1996.
55. D.J. Unwin. Geographical information systems and the problem of error and uncertainty. *Progress in Human Geography*, 19(4):549–558, 1995.
56. US Department of Defense. *Interface standard for vector product format*, 1996. MIL-STD-2407.
57. F.J.M. van der Wel, R.M. Hootsmans, and F. Ormeling. Visualization of data quality. In A.M. MacEachren and D.R.F. Taylor, editors, *Visualization in modern cartography*, pages 313–331. Pergamon: Oxford, 1994.
58. H. Veregin. Data quality parameters. In P.A. Longley, M.F. Goodchild, D.J. Maguire, and D.W. Rhind, editors, *Geographical information systems*, volume 1, chapter 12, pages 177–189. Longman: Essex, second edition, 1999.
59. C.G. Wesseling and G.B.M. Heuvelink. Manipulating qualitative attribute accuracy in vector GIS. In *Proceedings Fourth European Conference on Geographical Information Systems*, volume 1, pages 675–684, 1993.
60. M.F. Worboys. A generic model for planar geographical objects. *International Journal of Geographical Information Systems*, 6(5):353–372, 1992.
61. M.F. Worboys. Object-oriented approaches to geo-referenced information. *International Journal of Geographical Information Systems*, 8(4):385–399, 1994.

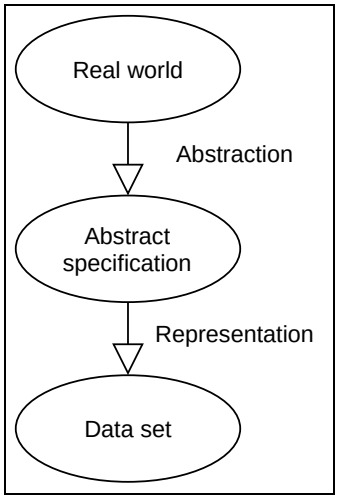

Figure 1. Conceptual model of IS

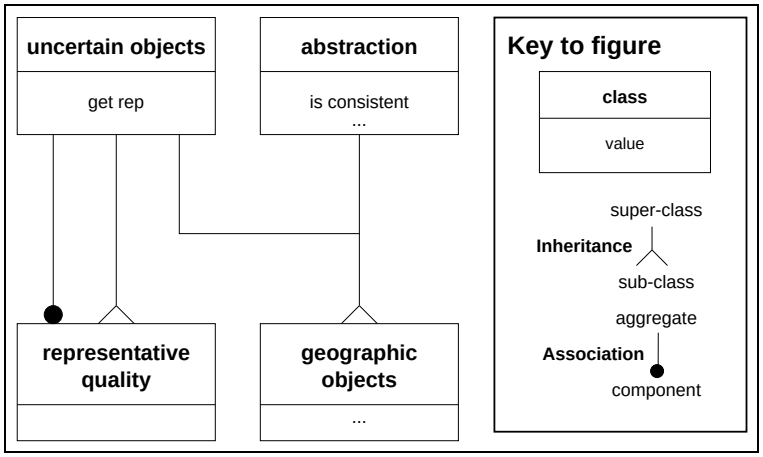

Figure 2. Simplified OOA results

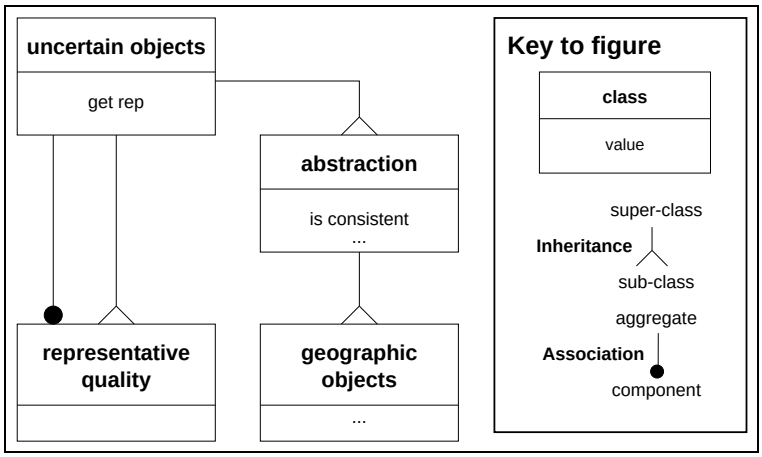

Figure 3. Single inheritance analysis results

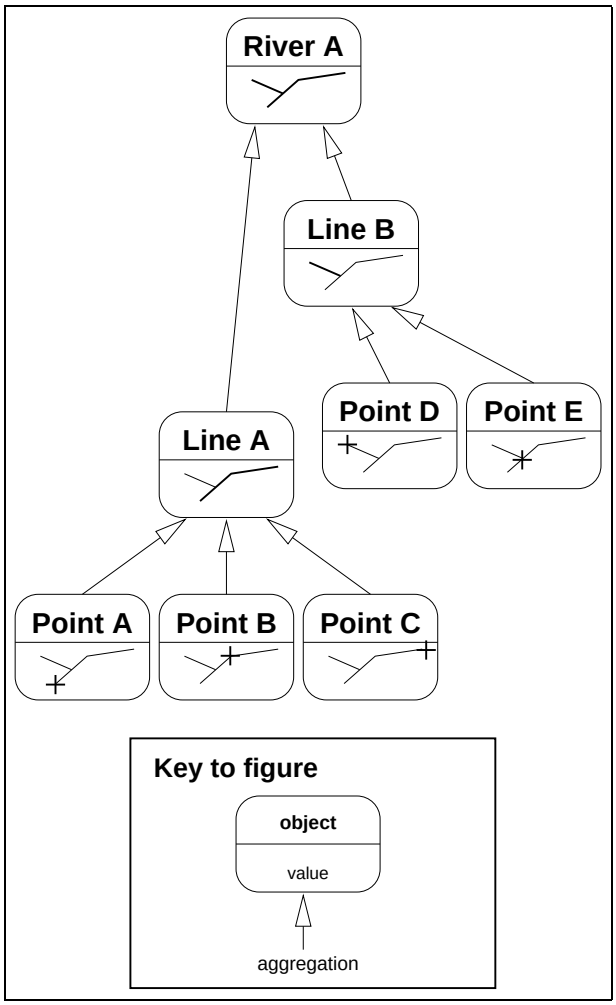

Figure 4. Aggregation in spatial databases

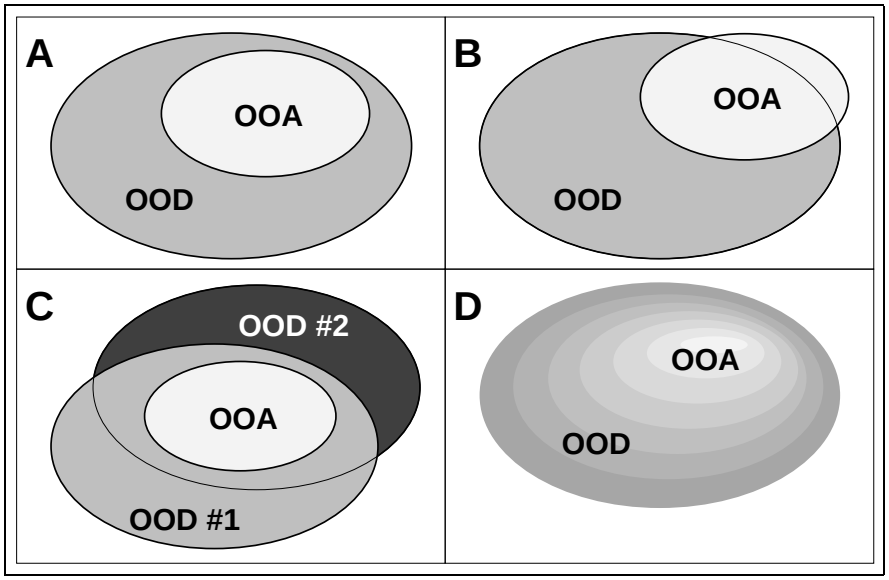

Figure 5. Summary of OOAD transition

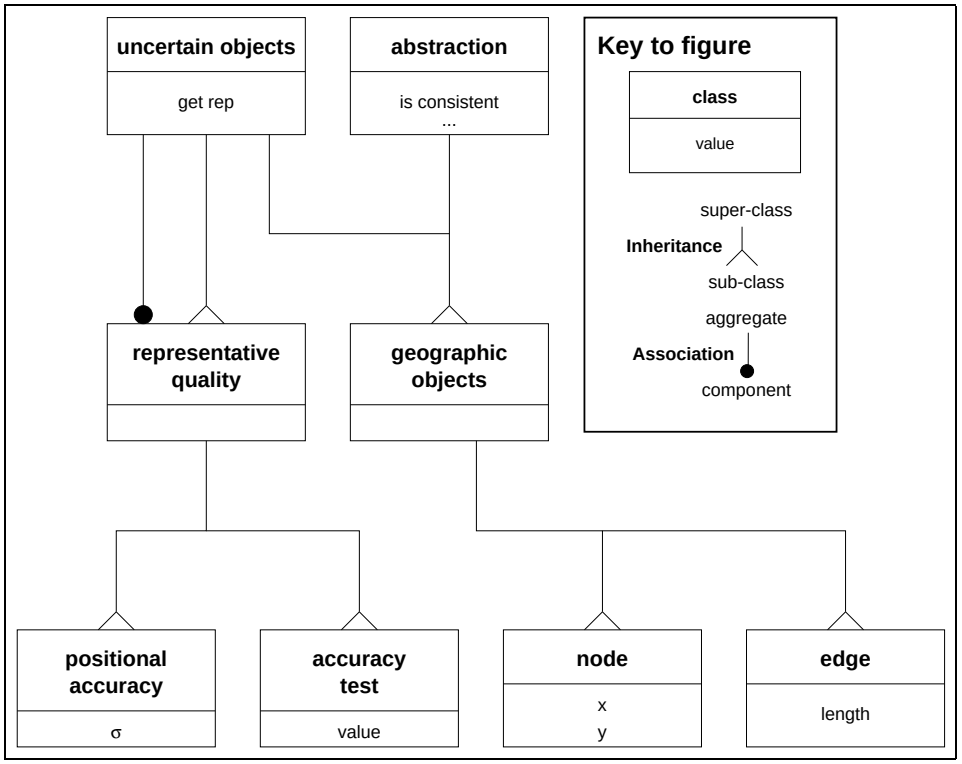

Figure 6. Example integrated error-sensitive schema