9-1988

Modeling Cadastral Spatial Relationships

Kenneth Dueker
Portland State University

Daniel Kjerne
Portland State University

ORIGINAL
DO NOT REMOVE!

PR 028

MODELING CADASTRAL SPATIAL RELATIONSHIPS

Kenneth J. Dueker
Daniel Kjerne

September, 1988

Center for Urban Studies
School of Urban and Public Affairs
Portland State University
Portland, OR 97207-0751

Dr. Dueker is Director of the Center for Urban Studies and Professor of Urban Studies and Planning, School of Urban and Public Affairs, Portland State University. Mr. Kjerne is Research Assistant, Center for Urban Studies, Portland State University.

This research was funded by a grant from the National Science Foundation.

APPENDIX VI

NATIONAL SCIENCE FOUNDATION Washington, D.C. 20550	FINAL PROJECT REPORT NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I—PROJECT IDENTIFICATION INFORMATION

1. Institution and Address Portland State University Portland, Or. 97207-0751	2. NSF Program Geography & Regional Science 4. Award Period From 8/1/87 To 1/31/89	3. NSF Award Number SES 871332 5. Cumulative Award Amount $60,000

6. Project Title

Modeling Cadastral Spatial Relationships

PART II—SUMMARY OF COMPLETED PROJECT *(FOR PUBLIC USE)*

Research exploring the object-oriented paradigm as a tool with which to model location of objects in maps is reported. The research led us into three areas:

- the use of an object-oriented language, Smalltalk-80 to model cadastral location using the object-oriented paradigm.
- study of the development of spatial information systems, such as engineering design and computer-aided design (CAD) database systems based on object-oriented ideas, to see how that approach might be useful for developing a mapping system.
- finally, the conceptual exploration of how an object-oriented approach might facilitate linkages of location among layers of a land information system.

Future research is needed to produce a more convincing prototype of a cadastral map. Since Smalltalk is a single-user, programming language environment, it does not produce objects that persist beyond the existence of the program in which they are created and manipulated. Computer-aided cadastral mapping requires an environment capable of working with persistent objects accessible by multiple users. This led to an investigation of present and future directions in database system development.

We see a productive course in continuing research on the object-oriented programming language prototype, adding more kinds of locational methods, improving the interface, and studying alternative structures to the Dictionary-based CadastralMap. Meanwhile, by keeping an eye on the development of object-oriented database systems, it should be possible to find one that can be made to manipulate spatial information. Finally, we hope to continue exploration of how the object-oriented paradigm will be useful in mapping and geographic analysis generally.

While the research reported here has emphasized cadastral objects, the paradigm for locational determination has a broad potential for application. Shared or common boundaries among layers or data types can be addressed by an object-oriented database approach, which lends itself to better handling of the interrelationships among objects of different types.

PART III—TECHNICAL INFORMATION *(FOR PROGRAM MANAGEMENT USES)*

1. ITEM *(Check appropriate blocks)*	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM: Check (✓)	TO BE FURNISHED SEPARATELY TO PROGRAM: Approx. Date
a. Abstracts of Theses		✓			
b. Publication Citations		✓			
c. Data on Scientific Collaborators	✓				
d. Information on Inventions	✓				
e. Technical Description of Project and Results	▨				
f. Other *(specify)*	▨				

2. Principal Investigator/Project Director Name *(Typed)* Kenneth J. Dueker	3. Principal Investigator/Project Director Signature Kenneth J Dueker	4. Date 1/10/89

NSF Form 98A (5-78) Supersedes All Previous Editions

Form Approved OMB No. 99R0013

INTRODUCTION

The work reported here follows on some previously described at several conferences (including the 1986 and 1988 Urban and Regional Information Systems Association [URISA] meetings and GIS/LIS'88) and available in published proceedings ([Kjerne and Dueker 1986], [Kjerne 1986], [Kjerne 1987]) exploring the object-oriented paradigm as a tool with which to model location of objects in maps. In the proposal to fund our research, we envisioned an ambitious program making use of what were anticipated to be existing capabilities to extend the storage of knowledge about the location of cadastral entities in a mapping database. Along the way, we planned to derive rules which could be used in future work to develop a cadastral mapping expert system. As we stated in the first pages of our proposal,

> The proposed research explores the capability of a new computer language paradigm to store the deep structural/spatial relationships among cadastral objects—relationships which are not modeled by present computer-aided mapping systems. Specifically, this research builds on promising exploratory work on the use of an object-oriented language to facilitate updating the cadastral layer. That work will be extended by use of an object-oriented geographic information system (GIS) to investigate the knowledge domain of the cadastral cartographer. Rules derived as a result of this investigation may be used in the construction of a rule-based cadastral mapping system. The present research will extend the use of the coordinate geometry approach to building a cadastral layer by incorporating the rules of evidence and procedures for locating property ownership from legal descriptions and property surveys in the database. [Dueker and Kjerne, 1987]

When it appeared that the development platform on which we had based this proposal would not, after all, be available in a timeframe or in a form that would be useful to us, we were forced to re-examine our capabilities and objectives. Our expertise, such as it was, lay in the subject field of cadastral mapping and system analysis rather than software development; we had no interest in building a mapping system from the ground up. Nevertheless, the object-oriented paradigm looked to be a powerful tool with which to explore map modeling, and we wanted to continue our emphasis on this question as well as on questions of implementation and utility of object-oriented design as a general contribution to computer-aided mapping and geographic information systems. Thus, our interests, abilities, and resources led us into three areas:

- the use of an object-oriented language, Smalltalk-80 (referred to hereinafter as Smalltalk), to model cadastral location using the object-oriented paradigm.
- study of the development of spatial information systems, such as engineering design and computer-aided design (CAD) database systems based on object-oriented ideas, to see how that approach might be useful for developing a mapping system.
- finally, the conceptual exploration of how an object-oriented approach might facilitate linkages of location among layers of a land information system.

The remainder of this report details the results of our invstigations into these three areas of inquiry. In addition, Appendix A lists Smalltalk object classes developed (these are also available as source code on diskette, along with a documentation file); Appendix B is a short disquisition on the effect of the object-oriented paradigm on computer mapping concepts.

MODELING CADASTRAL LOCATION

Principles of object-oriented systems

Object-oriented (o-o) systems are based on a paradigm of *objects* responding to *messages*, rather than (as is the case with procedural languages) on one of *operators* performing actions on *operands* [Goldberg 1983]. Here is one of the most succinct and comprehensive explanations of this environment that we have found:

> In an object-oriented environment, anything which is to be represented within the system is an object. An object consists of a private memory with a public interface. Each object within the system is identified by a unique system-defined object identifier. This identifier does not change even though properties of the object may change.
>
> An object's private memory cannot be directly accessed or manipulated by other objects. Any operation upon an object's private memory can only be effected by sending an appropriate request to the object. Since each object has a well-defined interface, and sole control over its own private memory, an object actually functions as an abstract data type.
>
> Many objects within the system exhibit similar behavior. These objects are grouped together in classes. Each object within a group is said to be an instance of the class of that group. Classes are arranged in a superclass/subclass hierarchy. The behavior defined within each class is inherited by its subclasses.
>
> Each class defines the behavior of its instances. However, the instances are not identical. Instances are distinguished by their unique identifiers and the contents of their private memories. The set of classes is not fixed. Users can create new classes as well as new instances of current classes. [Ketabchi and Wiens 1987]

It might only be added that, instead of classes, some object-oriented languages use a prototyping mechanism, cloning and modifying instances to add new types of objects [Ungar and Smith 1987]. And it should be noted that the description above is not completely true for procedural languages that have been extended to handle abstract data types, such as C++, Object Pascal, etc.

The o-o paradigm and conceptual modeling

Object-oriented languages are based on a very different way of viewing the world from that of procedural languages. In designing an application using a procedural language, we ask questions such as: What input do I have? What do I do to it to get the output I want? When using an o-o language, the questions become: What things are there in this problem? How do they behave?

This shift in viewpoint facilitates the transition from a conceptual model of a problem to a running application. If our model consists of things that interact in some fashion, an object-oriented application can have the same structure.

In the case of cadastral maps, we are concerned with a set of entities in a hierarchy of locational dependence (Figure 1).

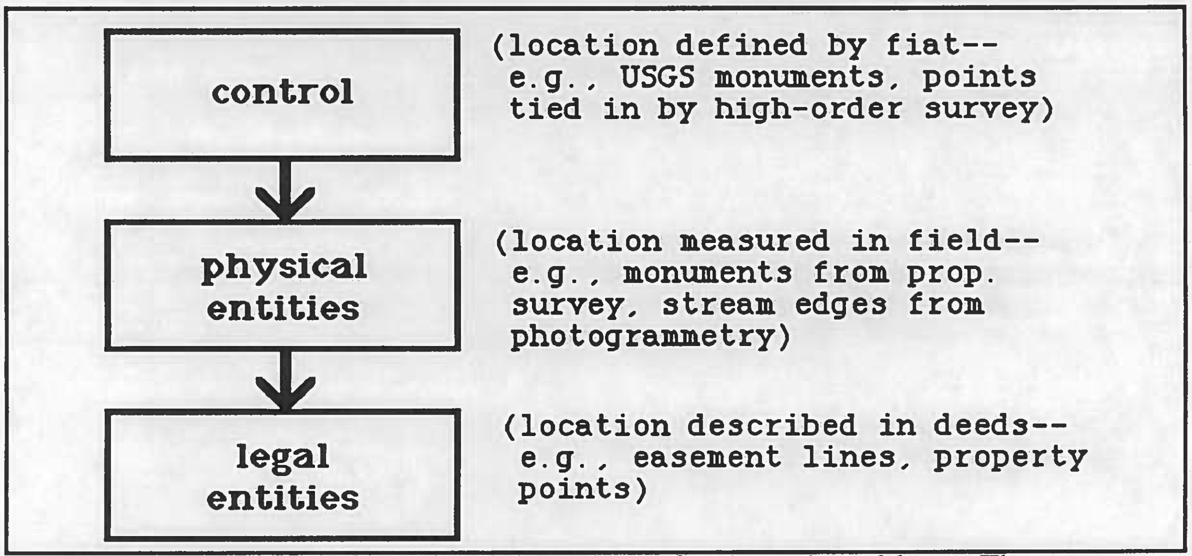
control
(location defined by fiat-- e.g., USGS monuments, points tied in by high-order survey)
physical entities
(location measured in field-- e.g., monuments from prop. survey, stream edges from photogrammetry)
legal entities
(location described in deeds-- e.g., easement lines, property points)

Figure 1. Hierarchy of locational dependency for the cadastral layer. The arrows may be read, "(entities at head) depend on (entities at tail) for location." Physical entities whose position has been located in the field are held steady; legal (non-physical) entities are plotted with relation to them. Among themselves, the locational hierarchy of legal objects follows a different, though still formalizable, set of rules than obtains for physical objects.

Except for control entities, everything knows where it is by applying a specific locational procedure (from surveying or legal description), with relation to a specific set of reference entities, and using specific measurement values.

In producing a map from a cadastre, we can take advantage of a number of simplifying assumptions since such a map is not a legally binding document in the sense that deeds are. We can assume, for instance, that an object will not occupy more than one location at a time, and that a single chain of location dependency can be determined for any object. (This does not mean that any object depends only on one other object as a reference object. In the case of closed traverses between fixed points, for instance, both ends of the traverse are reference points; objects in the traverse are located using a transformation procedure allowing the relative coordinates (and relative locations) to remain invariant, while the global position may change. Obviously if the *relative* positions of the endpoints changes, the traverse needs to be readjusted.)

Present status

Our previous work may briefly be described as involving the definition of object classes using a Forth- and Smalltalk-descended language called Neon. The present work has been carried out on an Apple Macintosh II in ParcPlace's Smalltalk-80 development environment for three major reasons: Smalltalk is a more powerful language; it is used more widely within the community of o-o researchers and developers; and it is easily portable to different applications and platforms.

A standard Smalltalk application consists of three parts (Figure 2):

- a Model, which can be any kind of object;
- one or more Views, each of which displays itself on the screen, making use of information contained in messages sent it by the Model;

- and a Controller, corresponding to each View, that accepts input from the user (via keyboard and cursor movement).

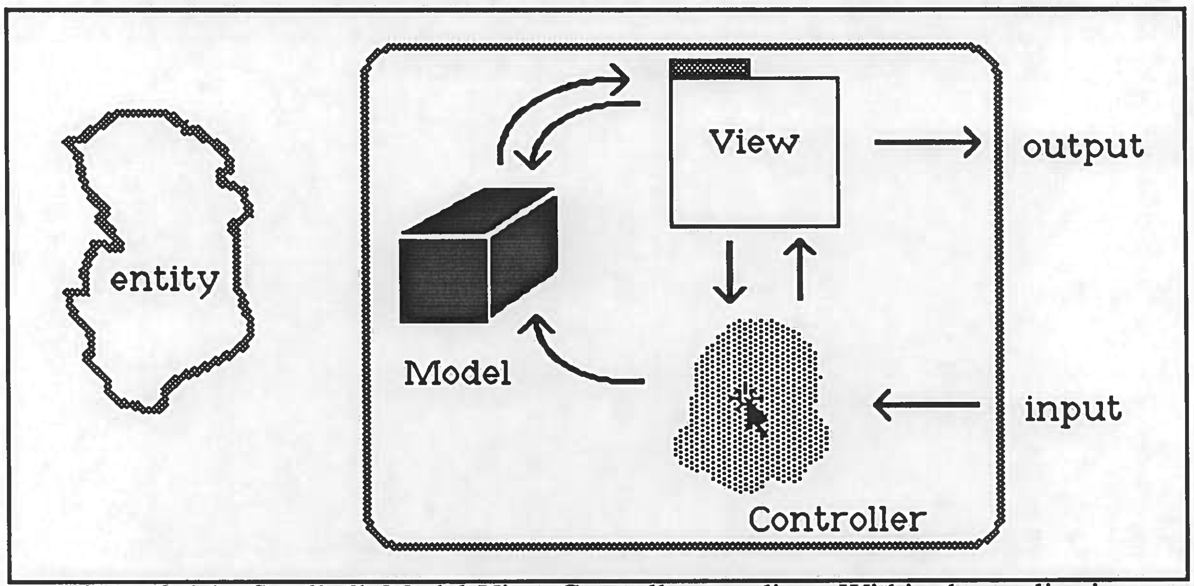

Figure 2. The Smalltalk Model-View-Controller paradigm. Within the application (gray outline), the Controller receives input from the user and typically sends a message to the View. The View in turn usually sends a message to the Model, which performs some action and replies to the View, which probably uses it for output. The View also sends messages to the Controller now and then. Less often, the Controller may have something to say directly to the Model. Meanwhile, out in the real world, is the entity being modeled. ("The map is not the territory." [Korzybski 1941])

All parts of the application are linked to some degree, but the Model is relatively independent of the other two. It doesn't really know, or care, how many Views are looking at it; all it does is respond to messages. The View, on the other hand, is tightly linked to a particular Model, and often performs some action automatically (such as redisplaying itself) if the Model changes. Similarly, the Controller and View are linked to each other so that the View (often) knows that it should perform some action when the cursor is within its window and a particular mouse button is down.

In the following discussion, we first describe the Model (and the objects it contains), the View, and then the Controller for this particular application.

Model (Cadastre). The Model in the prototype is a Cadastre (Figure 3), a kind of object that contains other objects in a list accessible by names, or keys (it is a subclass of a standard Smalltalk class called Dictionary.) As a subclass of Dictionary, it knows (among other things) how to add a new object to itself, how to tell how many objects it contains, and how to return a particular object when given the object's name. As a Cadastre, it knows in addition how to find the object within itself that is closest to a given point.

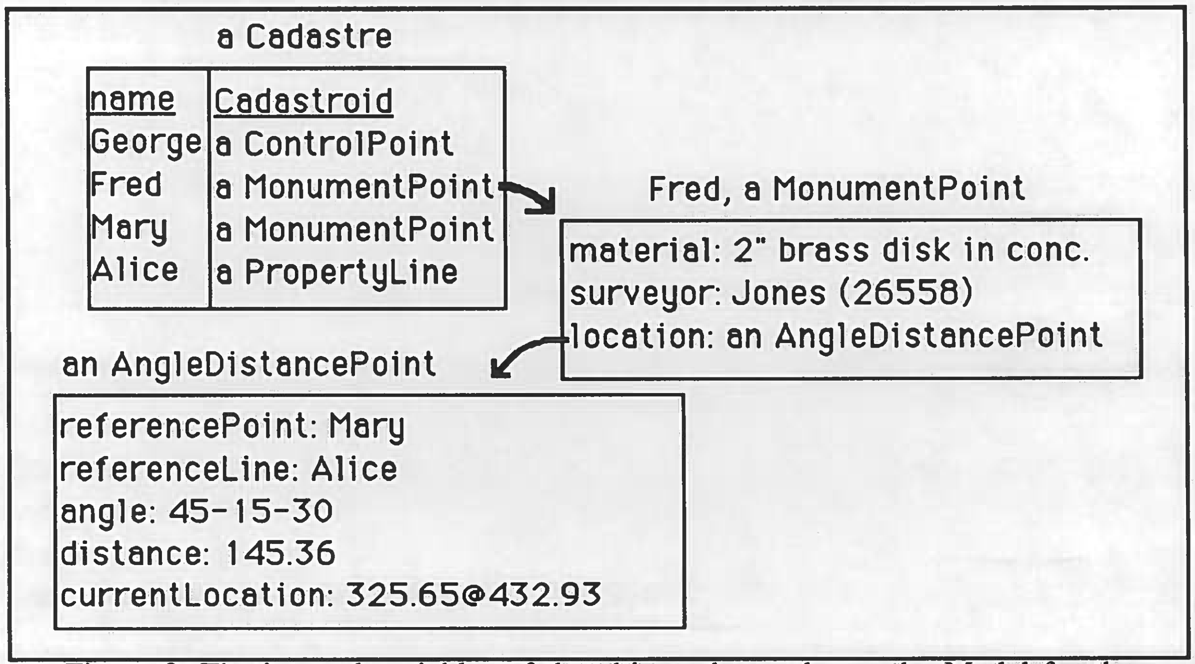

Figure 3. The internal variables of the objects that make up the Model for the prototype cadastral mapping application. A Cadastre contains a Dictionary-like list of Cadastroids, each of which is referenced by name. Each Cadastroid, in turn, contains a Locoid, which stores names of reference objects and values, and uses them to recalculate current location.

The objects in the Cadastre are Cadastroids. More precisely, they are all descended from the abstract class Cadastroid. (An abstract class is one which does not itself have instances, but is used to define behavior common to its subclasses.) The subclasses of Cadastroid are: MonumentPoint, ControlPoint, PropertyPoint, MonumentLine, and PropertyLine. As their names imply, these classes correspond to the kinds of things that appear on cadastral maps (aside from such things as grid ticks, legends, titles, and text).

Any object descended from Cadastroid has an internal variable called "location." An internal variable is part of the private memory of an object, and can itself contain an object. In this case, the "location" internal variable contains a Locoid. (Again, to be precise, Locoid is another abstract class, so the internal variable actually contains an instance of some subclass of Locoid.)

These are the objects that do all the work. Each subclass of Locoid corresponds to a locational procedure (AngleDistancePoint, AzimuthDistancePoint, AtPointPoint, IntersectionPoint, and StraightLine have been defined so far.) The procedure is used in one of the methods that class knows how to perform; it typically consists of asking the reference objects for their current location and applying values stored in other internal variables. Thus, for instance, an AngleDistancePoint object stores the name of two reference Cadastroid objects (one linear, from which the angle was measured, and one punctual, from which a distance was measured—usually, but not necessarily, at the origin of the line), and two measurement values (an angle and a distance). Computation of current location is a straightforward application of trigonometry.

In addition to capturing the "why" of where an object is positioned, a Locoid captures where it presently is: the current location is stored as a Point (x-y coordinates) or Line (OrderedCollection of Points) in an internal variable.

Other methods Locoids can perform include the ability to set up dependency relations with reference objects so that when one of these changes its location the dependent objects will be notified. Because of the dependency link set up between objects and their reference objects, locational changes propagate through the model. Thus, whenever a Cadastroid receives a message from one of its reference objects that its location has changed, its Locoid performs the locational procedure, the new current location is stored, and dependent Cadastroids are informed that a change has occurred.

View (CadastralView). The CadastralView is really, at this point, not a very exciting variation on the standard Smalltalk View. It is a single-paned window with the capability, when it is started up, of linking itself with a given Cadastre. It can send the Cadastre messages to have its elements display themselves in the CadastralView.

Controller (CadastralController). The CadastralController is also pretty rudimentary at this point. It can do one interesting thing: send a message to the Cadastre, along with the coordinates of a mouse-click, that will cause the Cadastre to return the name for one of its elements so that a window can be opened showing its internal variables.

Planned Near-Term Enhancements

More kinds of objects in Cadastre. More locational kinds of classes are needed before a reasonable property map prototype can be modeled. These include arc, spiral, and curve and more interesting (and robust) line intersection classes. Smalltalk does not have a generous supply of geometric classes; it is necessary to translate code from other languages or algorithmic descriptions. ParcPlace's new version of Smalltalk allows user-defined primitives [Smalltalk 1988]. This might be useful for importing code written in other languages to perform basic jobs such as computing intersections of lines, chains, arcs, and so forth.

In addition, it will be desirable to implement topological knowledge; this will require amendments to the object hierarchy. Topological objects ("Topoids"?) should be defined to go into a new internal variable slot of Cadastroids. They will require an ability to track dependency relationships similar to Locoids. For instance, the instantiation of a new area object will require methods to notify related line and point objects of the change.

Improvements to Cadastre. The choice of a subclass of Dictionary as the data structure for a Cadastre is still not fixed. It might be better to organize the assessor objects into another kind of Smalltalk Collection that could utilize efficient spatial search methods on object keys. The virtue of the present arrangement is that it is accessible using standard system tools, and it is reasonably close to the structures used in at least one object-oriented database system modeled on Smalltalk [Penney and Stein 1987].

Improvements to CadastralMap. To a surveyor or cartographer, the most serious deficiency at this point is that the map is upside down: y-coordinate values increase toward the bottom of the screen, in line with computer graphic convention. This is (in principle) not difficult to fix. Other possible improvements are cosmetic in nature: the addition of grid ticks, coordinate value text, perhaps a fixed window size and aspect ratio so that a given scale can be presented.

Improvements to CadastralController. At present, most interaction with the Cadastre is through the text workspace, a standard Smalltalk system feature. It would be nice to be able to add or edit points by selecting them with the cursor and working through a series of templates or dialog boxes. This is a lower priority than other goals, as it may be advantageous to use a commercially available application interface developer.

These improvements will eventually produce a more convincing prototype of a cadastral map; however, since Smalltalk is a single-user, programming language environment, it does not produce objects that persist beyond the existence of the program in which they are created and manipulated. Computer-aided cadastral mapping requires an environment capable of working with persistent objects accessible by multiple users. This leads us to an investigation of present and future directions in database system development.

MAPPING AND THE NEXT GENERATION IN DATABASE SYSTEMS

CAD and Mapping Environments

Practitioners involved in efforts to improve database systems are concerned with the question of modeling structures more complex than the typical business applications, and see their challenges in such applications as design engineering (especially mechanical and electrical), document authoring, and hypermedia. They do not see mapping as a high priority, but the problems attacked seem to us similar, especially to mapping of the cadastral variety.

It was particularly fascinating to read a description of the CAD application environment, substituting the words "cadastral cartographer" for "design engineer", "cadastral map" for "design object", and "update" for "refine":

> ...In these applications large amounts of data are created and shared by design engineers who do not want to concern themselves with the details of storage and data organizations....
>
> Design is the work of a team. Each member of the design team may begin a transaction which involves a large volume of data and persists for a long time. Therefore, conventional locking and time-stamping techniques cannot be used to control concurrent operations.
>
> Design starts with high level descriptions of a design object and continues through its iterative refinements. Since the history of the design operations must be maintained, refining a description should not destroy the original description, but must create a new description of the design object....[Ketabchi and Wiens 1987]

Each of the points raised—large data volumes, multiple users (with skills in a subject field other than data management), long transactions, necessity to track previous states of the database—is also characteristic of cadastral mapping.

Object-oriented Database Systems

We examined various candidate technologies proposed as a basis for advanced database capabilities—extensible toolkits (e.g., Exodus [Carey and DeWitt 1987]), semantic database systems (e.g., SIM [Tolbert 1988]), enhanced relational systems (e.g., POSTGRES [Stonebraker and Rowe 1986]), object-oriented (e.g. GemStone, Encore [Smith and Zdonik 1987], VBASE [Andrews and Harris 1987])—with two requirements in mind.

- First, as we have conceptualized the problem domain, we find we are dealing with composite entities: maps contain tracts, tracts or subdivisions contain parcels, parcels contain boundaries, and so forth. And the objects defined in our prototype cadastral location application contain location objects (which in turn contain current coordinate objects) and would presumably contain topological or non-geometric attribute objects.

- Secondly, we wish (with the inclusion of measurement procedures) to include behavior, and hope to be able to update location or topology "inside the database."

The object-oriented database approach seems more suited to meeting these requirements than any of the other general ones noted above.

Even within the area of research into object-oriented databases, it's still a fluxful situation, with a number of competing approaches. The main division seems to be between object-oriented database systems (oodbs) and object-oriented database programming languages (oodbpl).

An oodbs can manage composite objects and encapsulates behavior with values, but it is separate from the language interface used by the database's developers and users.

GemStone, for instance, runs as a "back end" on a minicomputer and responds to "front end" applications written in Smalltalk, C, or Pascal on microcomputers linked by a network [Purdy, Schuchardt, and Maier 1987]. An object-oriented database programming language would bring these two environments together. This approach avoids "impedance mismatch" between programing language (front end) and data manipulation language (back end) [Maier and Price 1984].

Within both oodbs and oodpl approaches, methods can be added to object classes (or, as database researchers are prone to call them, "types") to allow the kinds of things we are interested in for cadastral mapping: i.e., dynamic updating within the database by recalculating location, topological connections, and so forth. But it would appear to be advantageous to be able to import code from other languages as user-defined primitives that can be used inside methods. This would lessen the necessity to recast every spatial analysis method into a new language, since there is a large repertoire of functions, procedures, and techniques for spatial manipulation already extant in other languages, such as C.

Because of the inheritance of behavior, it is easy to extend object-oriented languages to have new capabilities. Research in progress has brought out extensions to Smalltalk that allow persistent objects [Kaehler and Krasner 1983] and a distributed Smalltalk [Bennett 1987]. To develop an oodbpl from this language, the next step would presumably be a distributed Smalltalk with persistent objects. But there appear to be problems with the class construct of Smalltalk when extended to persistent objects accessible by multiple users. All object-oriented languages don't implement characteristics peculiar to the o-o paradigm (such as inheritance) in the same way, though. A language that uses another mechanism (e.g., prototyping) may be found to be more appropriate as a basis for an oodbpl [Merrow and Laursen 1987]. Regardless of how that avenue of research turns out, it seems fruitful to continue using Smalltalk to develop more kinds of locational methods, since the basic types developed would be useful regardless of the particular inheritance mechanism.

The field of object-oriented database research is experiencing somewhat the same confusion and excitement that prevailed in GIS development a couple of years ago. As with GIS, the excitement and confusion in research spills over into marketing, and we have noted a number of claims among GIS vendors that their databases are “object-oriented” or “object-centered.” One of our frustrations in working on this project has been determining the precise meaning of these claims—whether the system in question will actually do what we think it should do, or whether, as seems frequently to be the case, the terms are used in a more limited sense. The difficulty in making these determinations is increased by the proprietary nature of most of these developments, so that between non-disclosure agreements, understandings that some things are not to be mentioned, and a general reluctance to go into detail, it’s just plain hard to tell what’s going on.

At present, we see the most productive course in following this line of research to be continuing to work on the object-oriented programming language prototype, adding more kinds of locational methods, improving the interface, and studying alternative structures to the Dictionary-based CadastralMap. Meanwhile, by keeping an eye on the development of object-oriented database systems, it should be possible to find one that can be made to manipulate spatial information. Finally, we hope to continue exploration of how the object-oriented paradigm will be useful in mapping and geographic analysis generally. The following section presents some thoughts we have already had on the subject.

EXTENSIONS TO MULTI-LAYER SYSTEMS

While the research reported here has emphasized cadastral objects, the paradigm for locational determination has a broad potential for application. Shared or common boundaries among layers or data types can be addressed by an object-oriented database approach, which lends itself to better handling of the interrelationships among objects of different types.

Currently, the representation of various themes or data types as separate layers in a GIS results in duplicate approximations of common boundaries. Some systems use a "shared primitive" approach to resolve this problem [Charlwood, Moon, and Tulip 1987]. In a relational model this requires a single underlying geometric layer, which is difficult to modify as changes may warrant.

For example, in a conventional GIS the cadastral layer may actually consist of sublayers of control, property lines, rights-of-way, etc. Yet, a right-of-way line usually is locationally related to a set of property lines, both of which depend on physical monuments (Figure 4). Depending on the particular situation (platted subdivision v. deed right-of-way, for instance), the controlling entities for location may be the lot monuments or other (centerline) monuments; the right-of-way may determine the location of property lines or vice versa. An object-oriented approach would allow modeling these diverse situations.

These locational dependency relations are not unique to the cadastral layer, however. They exist in a variety of boundary relations, and just as with the right-of-way lines, the procedures and types of reference objects for location of boundary segments of the same kind can vary from instance to instance.

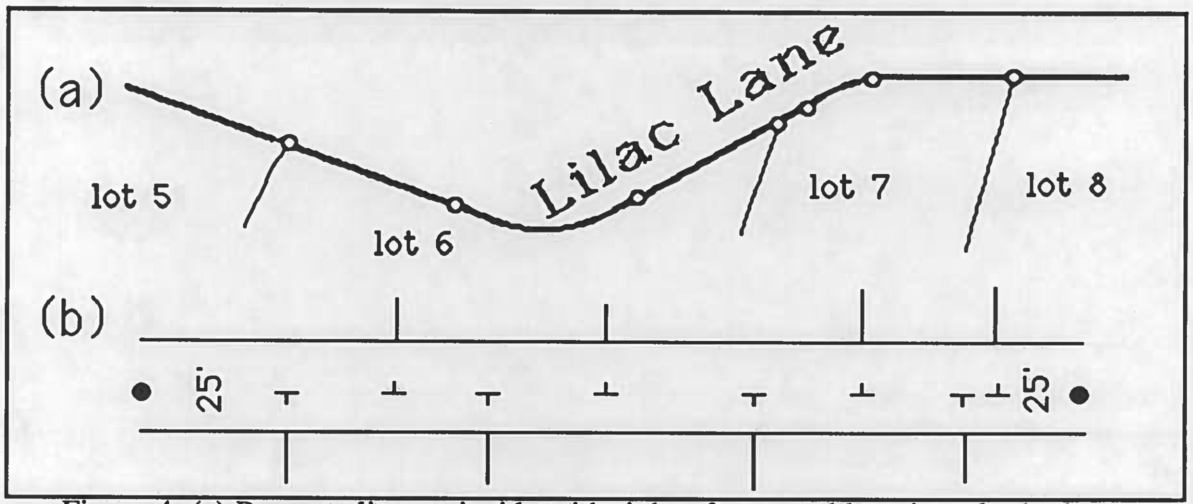

Figure 4. (a) Property lines coincide with right-of-way and locations for both are determined by parcel property monuments. (b) location for front lot lines is determined by location of right-of-way line, which is determined by location of centerline monuments.

For example, much of the northern boundary of the State of Oregon is the center of the main channel of the Columbia River, a natural feature. Segments of the state boundary are also county boundaries (Figure 5). The northern boundaries for these counties would be determined by a method contained within the county boundary object that would reference the state boundary, intersecting it with the east and west county boundaries. (The state boundary would locate itself by a method that referenced the channel of the Columbia River.) Meanwhile, the remaining county boundaries are probably monumented, referencing objects of a different class.

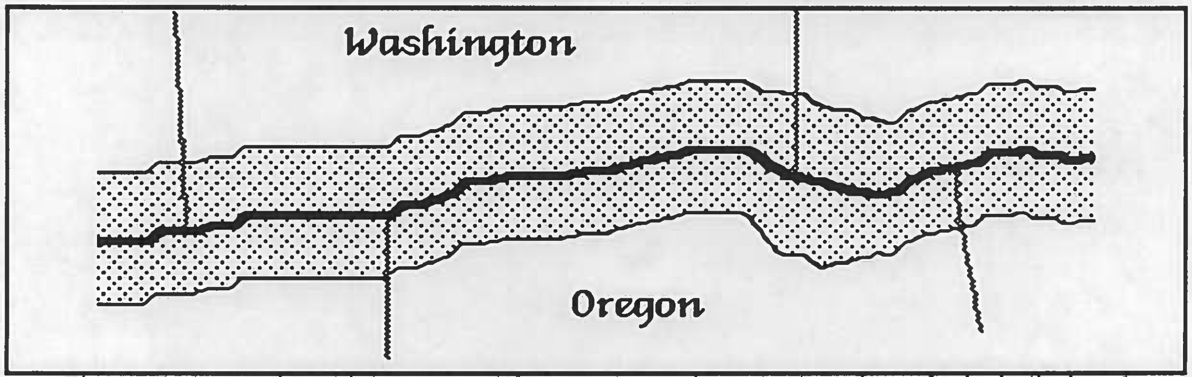

Figure 5. A case in which a natural feature determines the location of a jurisdictional boundary, with the locations of boundaries of subjurisdictions determined hierarchically. Location of the north-south trending county lines likely follows other procedures.

Users of multipurpose systems having a cadastral layer as a base are interested in four general classes of area entities: jurisdictions, statistical areas, natural resource areas, and property ownership areas. As with cadastral location, the boundaries of these areas depend for location on entities that have been located in the real world—on natural features, such as rivers or ridgelines, and on artificial features, in the form of streets, roads, and railroads. If this real-world structure is reflected in the model used for a mapping system, updating (inputting more accurate) locations of real-world entities will propagate those more accurate locations to dependents. This will result in more accurate responses to queries. Integrity of the database over time requires the capture in the data model of dependencies among entities.

CONCLUSION

We see the continuation of this work as involving two basic goals. The first (as stated in the introduction) is the development of a usable implementation of an object-oriented cadastral layer that can form one of the components of a land information system.

The second goal is of somewhat larger scope. It is to make use of the insights gained by the shift in perspective in moving from a procedural to an object-oriented approach in elucidating some fundamental questions about geographic information and our concepts of geographic space. Goodchild (1988) has already noted the need to escape traditional constraints of emulation of existing cartographic products in the transition from an analog to a digital mapping technology. More effective models of spatial relationships are needed. We anticipate that the shift from a procedural to an object-oriented approach, while not as fundamental, can nonetheless be valuable.

For example, Peuquet's (1988) initial presentation of a dual conceptual model of the representation of geographic space, while obviously making use of some of the concepts found in the object-oriented approach, can be further extended conceptually, as well as benefitting from use of object-oriented design in implementation.

To take the latter point first, one need only note that "locations" (e.g. grid cells)--the basic entities in the location-based representation phase of the dual model--can be implemented as objects in a computer implementation as readily as can the "objects" in the object-based representation phase. This unity of design would greatly facilitate the goal of transferring information from one phase to the other.

Two insights appear from consideration of Peuquet's model with the object-oriented paradigm in mind. The first is that, in addition to seeing a general correspondence between raster-based and vector-based data models in the two phases of the dual model, one may also see a correspondence to the procedural and object-oriented paradigms. This correspondence is obvious for the object-based representation, whereas for the location-based representation, it is sufficient to note that "locations" are pieces of data that are operated on by generalization, overlay, or other procedures to produce a new data set. Thus, in attempting to transfer information from one phase of the model to the other, we are faced not merely with the problem of translating the data format, but also the semantics--the meaning and structure--of the representation. Looking at things in this way casts a little doubt on the strict symmetry of the two phases of the data model as developed by Peuquet, and leads into the second insight: are there really only three possible kinds of relationships (taxonomic, thematic, and membership) in each phase of the model?

Possibly not. As we recall the tools available in each paradigm, or each phase of the model, to derive attributes and relationships, we recall that, for the location-based phase, knowledge resides in the procedures applied to the data; for the object-based phase, knowledge resides in the structure of the object classes <u>and in the methods understood by those classes</u> (not in external procedures). Thus, while Peuquet mentions inheritance and, implicitly, prototyping (or the class-instance mechanism) as important elements in defining thematic, taxonomic, and membership relationships, she does not mention the role that polymorphism or composit objects can play. As we have developed the prototype cadastral application in the present research, we have noted how often the "part-of" relation, implemented by the composite object mechanism, has been important in devleoping a model of a certain class of spatial object.

At this point, we do not know where these ideas will eventually lead in the development of a conceptual model of geographic space, but it is our expectation that the object-oriented paradigm will provide some help in forming the multiple representations of objects in