Object-Oriented Approaches in the Design of More Capable (Adjustment) Systems

Tapani Sarjakoski
Finnish Geodetic Institute
Ilmalankatu 1 A, SF-00240 Helsinki
Finland
Commission III

1 Introduction

Object-oriented is well on its way to becoming the buzzword of the 1980s. Suddenly everybody is using it, but with such a range of radically different meanings that no one seems to know exactly what the other is saying [Cox86]. In this paper we study object-oriented approaches, to be used in the design and implementation of systems. The presentation is given on a rather general level, with the purpose to show that there is a 'methodological synergy' among object-oriented approaches. By this we mean that various methods that have been introduced rather independently, rely on the use of a single central concept: *object*. In the evolution process these methods seem to converge. The applicability of object oriented-approaches is also wide: from database management systems to graphics-display applications. In addition, there is a clear tendency that the whole software lifecycle — from the requirement analysis to implementation and maintenance — can be supported by using object-oriented approaches.

The paper also gives an example of the object structure related to block adjustment, reflecting the author's interest and work in the development of more capable adjustment systems for photogrammetric purposes — in the sense that ever higher-level tasks are carried automatically. It is emphasized, however, that the techniques are general and it is likely that the greatest benefits of object-oriented approaches are likely to be encountered in application areas were the family of object classes is large and varying; a somewhat opposite situation is prevailing in adjustment applications.

2 Object-Oriented Approaches

Three views to object-oriented approaches are considered below: object-oriented programming, object-oriented databases, and object-oriented design. The purpose is to show that these views are very much interrelated, only the weighting of different aspects varies.

2.1 What is Object-Oriented Programming?

When speaking of object-oriented programming, a distinction must be made between object-oriented programming *paradigm* and object-oriented programming *language*. With the former we mean a programming 'style' or 'way of thinking': a problem to be solved is modeled in terms of *objects* and *relationships* between them. The latter, on the other hand, is such programming language which supports an implementation of a of a program that has been design using object-oriented paradigm.

Object-oriented programming paradigm. *Objects* are the very central element in object-oriented paradigm: they correspond to some 'real world' objects. In that sense object-oriented paradigm is based on *simulation principle*: the object-structure is constructed to model the situation in the real world. Objects have an *identity*, i.e. each object can be distinguished from all the others. An object retains its identity as long as it exists — although its properties might change.

Each object is an *instance* of some object *class*. The definition of a class fixes its characteristics: the *instance variables* or *attributes* and the *procedures* related to it. Procedures are usually considered to consist of a procedure body (*method*) and a specification for 'calling' a procedure, a (*protocol*). An actual call of a method is then realized by sending a *message* to a specific, identified object.

An object-oriented programming paradigm is also strongly based on the concept of *class-hierarchy*: object-classes are organized in a hierarchical manner, the most general classes being on the top of the hierarchy. The more specialized classes down in the hierarchy then *inherit* the characteristic of the ones upwards in the hierarchy and also have some additional or varying characterists, i.e, instance variables and methods. The protocol for similar object-classes in kept identical. This allowes similar objects to be treated in a uniform manner.

Object-oriented programming languages. Object-oriented programming language supports an implementation of a program design which has been based on an object-oriented paradigm. In the analysis of object-oriented languages several questions have to be considered:

1. Language itself
   (a) self-contained language
   (b) language extension
   (c) various language features
      i. dynamic binding
      ii. hierarchical class definitions
      iii. multiple hierarchies
2. Programming environment

(a) use of graphical CASE[1] tools

(b) interpretative environment

(c) editing - compilation - linking - running

(d) use of preprocessors

Implemented object-oriented programming systems are combinations with respect to the listed features [SB86], [Cox86]. Smalltalk implementations — Smalltalk being the most well-known and rather original object-oriented programming language and system — have the features 1ai, 1aii, 2a, 2b, which makes it especially suitable for experimental work and fast prototyping.

2.2 What is an Object-Oriented DBMS?

Database management system (DBMS) is an implementation of some specific data definition language and data manipulation language [Dat83]. The datamodel of an object-oriented DBMS is based on objects. The GemStone database system reported in [MS87] is an example of an object-oriented DBMS. To make it brief, GemStone system deviates from Smalltalk system only in one principal aspect: the objects are stored in a diskstorage which allows them to be 'long-living' or 'permanent'. GemStone database can thus be understood a snapshot of the state of a Smalltalk-like system. The three pricipal concepts of the GemStone model and language are *object*, *message*, and *class*. Figure 1 illustrates the relationship between object-oriented and conventional database systems.

The use of object-oriented database system can be advocated by the following arguments:

- There is no reason to 'flatten' the real-world object model to fit into the relational formalism.
- Compared to DBMS based on 'pure' relational formalism, there is need to include some computational power into DBMS.
- Some anomalies related to the use of relational formalism will be avoided.
- If program development is based on use of object-oriented approaches, it is likely that some problems due to use of different datamodels (called impedance mismatch in [MS87]) are avoided.

[1]Computer Aided Software Engineering

Approximate Equivalences

GemStone	Conventional
object	record instance, set instance
instance variable	field, attribute
instance variable constraint	field type, domain
message	procedure call
method	procedure body
class-defining object	record type, relation scheme
class hierarchy	database scheme
class instance	record instance, tuple
collection class	set, relation

Figure 1: The correspondence between object-oriented and conventional database system [MS87].

2.3 What is Object-Oriented Design?

Object-oriented programming and database management primarily deal with such *formal* notations that can be mechanically translated to computer programs. With object-oriented design, on the other hand, we mean the *human activities* that aim to produce object-oriented programs or database definitions. To support these activities, *formal* and *informal* methods have been developed.

Object-oriented database design. Object-oriented approaches have a long history in the design of databases, although varying terminology has been used. So-called *entity-attribute-relationship* (EAR) model is used frequently. EAR model is mainly used as an *intermediate* informal, graphics-based method to describe real-world entities,their properties and the relationships between them. The actual implementation, however, has been based on some of the well known database models: relational, hierarchical, or network.

Object-oriented program design. The work in [Boo86] is an example of object-oriented program design methods. According to [Boo86], object-oriented program design consists of the following steps:

1. Identify objects and their attributes
2. Identify the operations suffered by and required of each object
3. Establish visibility of each object in the relation to other objects
4. Establish the interface of each object
5. Implement each object

It is to be noticed that the method does not consider *class hierarchy*. This can be understood in the light of the target language which is in this case ADA. Much of the power of the object-oriented paradigm is lost, however.

Object-oriented system design. As seen above, object-oriented program systems and database systems can be considered to be identical, theoretically. In the sense of design, the most distinct difference between programs and databases system is that the latter are used (primarily) to store data while the former are used (primarily) for the manipulation of data.

Considering complete systems, both of these two aspects are present and a design method must address both. Jackson Structured Development (JSD) [Jac83] serves as an example of design methods that address these concerns. It models a system as a set of objects which are considered to be *long-running processes*. It also addresses the question of how objects are allowed to evolve in the time.

JSD has been developed to meet the needs encountered in the design of complete administrative systems. The development work also seems to have been independent of the one made by the 'object-oriented community'.

3 Objects in Photogrammetric Block Adjustment

The object-structure in a photogrammetric block adjustment task is briefly studied here, for illustrating how object-oriented approaches can be used in the analysis and design of a photogrammetric application programs. Figure 2 outlines the 'object world' related to bundle block adjustment [Sar88], EAR-diagrams are used. Main attributes of the objects are given in Table 1. The symbol G-E-D is used to denote that the corresponding observation(-object) is affected by a decision about the presence of a gross error.

The role and importance of making the object-structure explicit can be motivated by the following comments:

- More advanced adjustment techniques — compared to plain, numerical least squares adjustment — as detection and correction of gross errors, use of additional parameters to compensate for systematic image errors, variance component estimation, etc., require more information about the structure of observations.
- The functionality of the relationships (*one-to-one, many-one, ...*) can be used to as a tool to pose structural constraint about what is possible in reality. Thisway more comprehensive concistency-checks are possible.
- From the point of view of least squares adjustment, two main-classes of objects can be recognized: observations (photogrammetric obs., control-point obs., level constraint obs., ...) and groups of unknown parameters (points, photos, levels, groups of add. par). Each of these two classes have some common characteristics within its subclasses but also some subclass-specific characteristics.
- It is convenient to consider an adjustment task as a constraint-satisfaction problem: output attributes of the objects are constraint by the input attributes and the adjustment model.

Entity (Object)	Input-attributes	Output-attributes
Photo	—	XYZ, ω, ϕ, κ
Strip	—	—
Flight	—	Additional param. to compensate image deformations
Camera	—	—
Calibration data of camera lens	1) Camera constant 2) Coordinates of fiducials 3) Tables for radial distortion	—
Set-up	—	Parameters of camera coordinate transformation
Stereo model	—	—
Fiducial observation	Instrument coordinates, xy	Residual-vector from camera coordinate transformation G-E-D
Photo observation	Instrument coordinates, xy	Residual-vector from bundle block adjustment G-E-D
Object point	—	XYZ-coordinates
Object point class	*A priori* variance of related photogrammetric observations	Estimated variance component(s) of related photogrammetric observations
Datum point	XYZ-, XY-, or Z-coord.	Residual-vector G-E-D
Datum point class	*A priori* variance	Estimated variance component(s)
Level constraint	(Implicit zero height difference)	Z-residual G-E-D
Object level	—	Z-coordinate
Object level class	*A priori* variance of related level constraints	Estimated variance component of related level constraints
Datum level	Z-coordinate	Z-residual G-E-D
Datum level class	*A priori* variance	Estimated variance component

Table 1: Input and output attributes in the EAR-model for block adjustment.

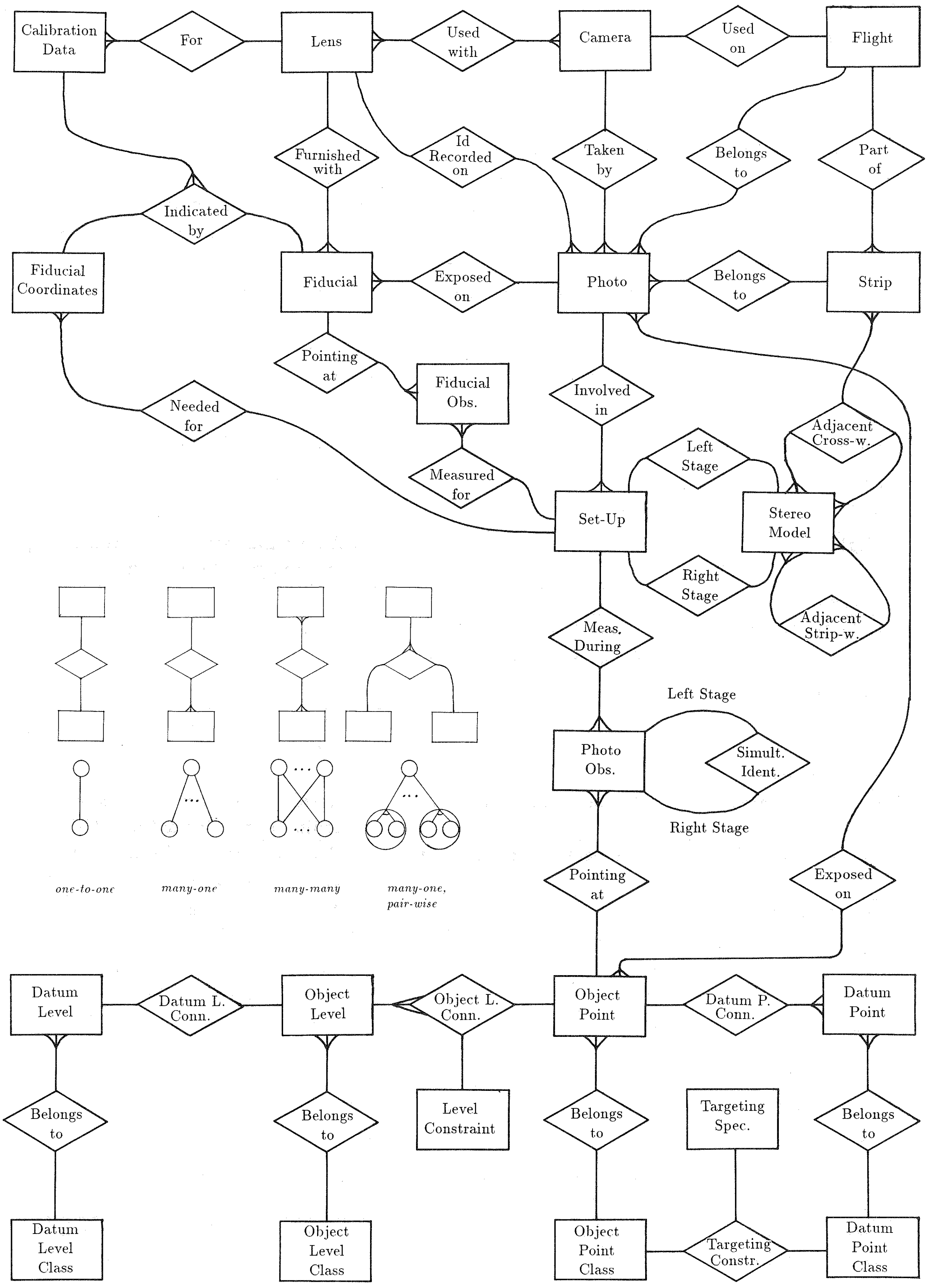

Figure 2: EAR-model for the data-entities involved in block adjustment.

4 Review of Literature

The title 'object-oriented approach' covers a wide range of research and literature. Above we have only pointed out subtopics that were considered to be most central. In the following some reference material is reviewed, to guide an interested reader into further studies.

[SB86] gives a introduction to the concepts of object-oriented programming, surveys some of the important variations and open issues of object-oriented programming.

[OOPSLA86], [SW87], and [ECOOP87] contain the papers of conferences of object oriented-programming (and databases). The great variety of issues relevant for object-oriented programming is striking.

[Boo86] discusses a method for object-oriented program development. The paper also examines the mapping of object-oriented techniques to ADA using a design case study. It is concluded that reusable software components tend to be objects or classes of objects.

In [Cox86] the main emphasize is in the introduction of *Objective C*, an (commercial) object-oriented extension to C-language. It also gives a rather good comparison of some object-oriented languages.

[Cam86] gives an overview of the Jackson System Development (JDS) method [Jac83]. JSD specifications consist mainly of a distributed network of processes that communicate by message passing and by read-only inspection of each other's data. A JSD specification is therefore directly executable, at least in principle. The paper does not relate JSD to object-oriented programming but the relationship can be recognized immediatelly.

[MS87] and [Ste88] study development and implementation of an object-oriented DBMS. The design is based on SMALLTALK-like concepts and syntax. Noticeably, the work is carried out by a commercial company.

Section 5.6 (Semantic data modeling) and Chapter 6 (The Extended Relational Model RM/T) of [Dat83] clearly show the connection between EAR-models and object oriented models. In addition, the shortcomings of pure relational formalism is discussed. It becomes apparent that more powerful notation (and more object-oriented) than the pure relational formalism is required to cope with many real-life situations.

The need of extensions to the EAR-model is recognized in [Mak84], [EWH85], and [Laz87], to mention some references. In all of them, as in [Kel86], the concern is also to include more 'meening' or semantic power into a system.

The author's earlier papers [Sar86] and [Sar87] are related to the current topic.

5 Concluding Remarks

We summarize the presentation with the following remarks.

- Active research and development work is in progress on developing tools to support object-oriented approaches:
  - Languages
  - Development aids for object-oriented programming: interactive environments, graphical tools
  - General-purpose object-oriented DBMS
- Development trends related to various object-oriented approaches are converging.
- DBMS based on object-oriented approach are promising.
- A database (system), as a whole, will be understood increasingly as a *process*, instead of a *variable*.
- It can be forseen that object-oriented approaches are not free of problems and heuristic decisions. For example, do we place context-dependent knowledge in the objects or in the context, i.e., in the program that manipulates the objects? Simulation principle can be used as a guide-line, i.e., a system should model real-world activities as closely as possibly, in structure and behaviour. Thus, the *notion* of message passing is not always realistic; only active objects are able to recieve messages.

Acknowledgement

The Academy of Finland has financed the research this report is part of. During the research the author has used the facilities of Technical Research Centre of Finland and Helsinki University of Technology.

References

[Boo86] Grady Booch. Object-oriented development. *IEEE Transactions on Software Engineering*, SE-12(2):211–221, 1986.

[Cam86] John. R. Cameron. An overview of JSD. *IEEE Transactions on Software Engineering*, SE-12(2):222–240, 1986.

[Cox86] Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach.* Addison-Wesley Publishing Company, 1986.

[Dat83] J.C. Date. *An Introduction to Database Systems, Volume II.* Addison-Wesley Publishing Company, 1983.

[ECOOP87] *European Conference on Object-Oriented Programming,* Conference Proceedings, June 15-17, 1987, Paris, France. ISSN 0221-5225.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: an extension to the entity-relationship model. *Data & Knowledge Engineering*, 1(1):75–116, 1985.

[Jac83] M.A. Jackson. *System Development.* Prentice-Hall, 1983.

[Kel86] Charles Kellogg. From data management to knowledge management. *Computer*, 19(1):75–84, 1986.

[Laz87] Rafael Lazimy. A generic shell approach for knowledge elicitation and representation in IDSS. In *Proceedings of the Eighth International Conference on Information Systems*, pages 335–350, Pittsburgh, Pennsylvania, 1987.

[Mak84] Kirsi Makkonen. *On Urban Land Information Systems — A Semantic Approach to Analysis and Design.* The Finnish Society of Surveying Sciences, Publication 17, Helsinki, 1984. Doctoral thesis.

[OOPSLA86] *OOPSLA'86, Object-Oriented Programming Systems, Languages and Applications,* Conference Proceedings, Sept. 29 - Oct. 2, 1986, Portland Oregon. Special Issue of ACM Sigplan Notices, Vol. 21, Nr. 11, 1986.

[MS87] David Maier and Jacob Stein. Development and implementation of an object-oriented DBMS. In Ref. [SW87], pages 355–392, 1987.

[SB86] Mark Stefik and Daniel G. Bobrow. Object-oriented programming:Themes and Variations. *AI Magazine,* 7(4):40–62,1986.

[Sar86] Tapani Sarjakoski. Software engineering in photogrammetric systems. In *International Archives of Photogrammetry and Remote Sensing,* Vol. 26–3/2, pages 588–601, Rovaniemi, 1986.

[Sar87] Tapani Sarjakoski. Artificial intelligence in photogrammetry. *Photogrammetria*, 42:245–270, 1987.

[Sar88] Tapani Sarjakoski. Automation in Photogrammetric Block Adjustment Systems — On the Role of Heuristic Information and Methods. *Acta Polytechnica Scandinavica, Ci88*, Helsinki, 1988. Doctoral thesis.

[Ste88] Jacob Stein. Object-oriented programming and database design. *Dr. Dobb's Journal of Software Tools*, 13(3):18–35, 1988.

[SW87] Bruce Shriver and Peter Wegner (Eds). *Research Directions in Object-Oriented Programming.* The MIT Press, Cambridge, Massachusetts, 1987.