December 1993

Parallel Modeling in a Multi-Windowing Enivronment

Gee-Kin Yeo
National University of Singapure

Parallel Modeling in a Multi-Windowing Environment

GEE KIN YEO, Department of Information Systems and Computer Science,
National University of Singapore,
Kent Ridge, Singapore 0511, Republic of Singapore

ABSTRACT

Parallel modeling has been proposed to support model-solver integration in decision support systems, particularly in performing sensitivity analysis. This paper first presents a multi-windowing environment that supports parallel modeling for production-transportation problems. Object-oriented concepts are then applied to illustrate possible extensions to include further modeling capabilities.

1. Introduction

In decision support systems, model formulation is concerned with the choice of a modeling paradigm and the construction of a formal model specification within that paradigm. Modeling languages have been developed to facilitate this formulation process. In addition, a modeling language system usually takes further steps into submitting the resulting model specification to a solving algorithm to obtain a solution. In this paper, a formulation in such a modeling language is termed as a solver representation. The solver language therefore refers to the modeling language used. In a typical decision making environment, modeling languages may be directly used by modelers or interfaces may be built to translate model specifications in still higher-level representation to a solver representation. Representations from these user interfaces are called the modeler representations. Ideally, solution from the solver should be included in the modeler representation when it is known.

The principal design objective of PTS - a graphical interface for Production and Transportation System [Kendrick, 90] - has been to provide visual access to parallel model representations of a general linear programming production and transportation system application. The model representations included initially are a graphical network showing the interconnections of plants and markets, tables of data values on production and transportation, and a solver language in textual display. The solver language referred to in PTS is GAMS [Brooke, 88].

The graphical and data views as in PTS can be considered as specifications of the modeler's representation and the GAMS view is the specification in the chosen solver representation. These two representations differ in content details and forms. The modeler's representation is used to specify the modeler's understanding of the problem and data during the modeling process. Since it should be in terms of constructs that are natural to the modeler, graphical network of system components and tables of data values are chosen as the forms of representation. The modeling language of GAMS and again tables of data values are the forms of representation for the solver.

There are several shortcomings in the initial version of PTS, one of which has been in its ability to support only representations of a single problem. Sensitivity analysis can therefore only be performed by comparying solver outputs outside the modeling environment. There is also no direct support of the creation and manipulation of the link between a plant and a market in the modeler representation. All the plants and markets of a model are automatically connected, thus making it unrealistic when some links are infeasible in a problem. In the next section, we present a revised implementation of PTS that overcomes these shortcomings in a multi-windowing environment.

To support modeler-solver integration, it is important to have model consistency in the parallel model views; i.e., any modification made in one view should be reflected in the other view. However, in the initial implementation of PTS, only the modeler representation can be created graphically from scratch but the solver representation can only be created, although automatically, from existing modeler representation. In other words, the transformation from the solver representation to the graphical or data view in the modeler representation is not supported. Even using the present-day artificial intelligence technique, it is very difficult to allow a user to fully exploit the powerful modeling capabilities of a solver language such as GAMS and then automatically transforms the program correctly into a modeler representation such as a PTS graph. Within production transportation problems, it is easier to approach the problem by identifying as many different types of models as possible, viz., single-commodity or multiple-commodity, inclusion or non-inclusion of production cost in the transportation cost, etc., and build a model base with the generic forms or templates of the solver representations of these models. Transformation algorithms can then be applied to obtain the modeler representations. Solver language modeling would then become only instantiation of these templates by supplying object names, values and at most computation formula.

In section 3, we illustrate our initial attempt in the possible extensions in parallel modeling using an object-oriented approach, which has been acclaimed for its provision for extensibility and reusability.

2. A Multi-windowing Modeling Environment

Our version of PTS runs under Microsoft Windows™ and supports MDI, or Multiple Document Interface, which is an interface standard for Windows applications that allows the user to simultaneously work with many open documents. By allowing users to tile, cascade, maximize and minimize documents with an MDI, users can easily compare different document contents easily. As shown in Figure 1, four different documents pertaining to the same problem in PTS are displayed: two of graphical type showing the model representations, the original formulation (*.DAT) and the optimal solution (*.RST); another two of text type showing the solver representations, the GAMS program (*.GMS) and the result listing (*.LST). The documents are displayed in 'child' windows, all open in a 'client' window that serves only as a work space. 'Child' windows have no menu of their own. The menu on the 'client' window applies to the 'child' document window that is active at any given time. In Figure 1, TRNSP.DAT is the active window and one of the menu item SOLVE actually allows the user to activate the conversion to solver representation or to compile the converted GAMS program. All the document child windows are clipped to the workspace area and never appear outside the 'client' window. An MDI offers the advantage of maintaining a document list throughout the operation of the application. Selection can be made at any one time from this list to make any document window active. Scrolling can be used to locate particular points of interest in any document. Cut-and-paste commands for editing in text mode for the solver representations and in draw mode for model representations will be made available in the next version. Thus, it will be possible to import a GAMS program from a text editor or to transfer data from other applications such as spreadsheets or databases into databoxes of the model representations.

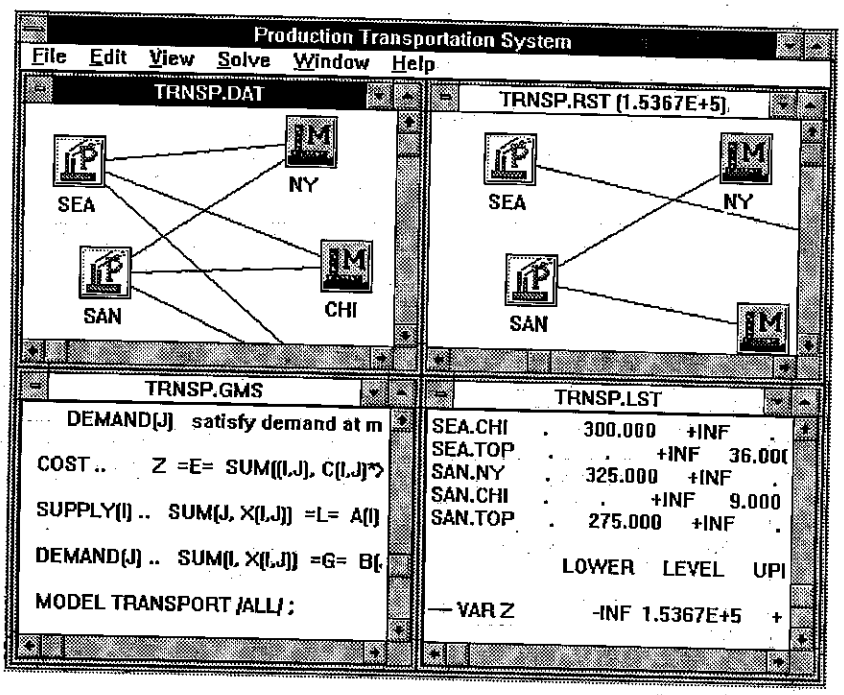

Figure 1

Initially, the menu in the opening window as shown in Figure 2 offers option to create a new document or open an existing document. To create a graphical model representation, a user chooses appropriate buttons to draw plant, market or link symbols appreaing in a ribbon control window as shown in Figure 3. The user is free to reposition any symbol by simple clicking and dragging. Data associated with each symbol can be entered into databoxes opened up by double clicking on the symbols. A link databox appears in Figure 4. After the data associated with plants, markets or links become available, the user can turn on/off option to display or hide them, as shown in Figure 5.

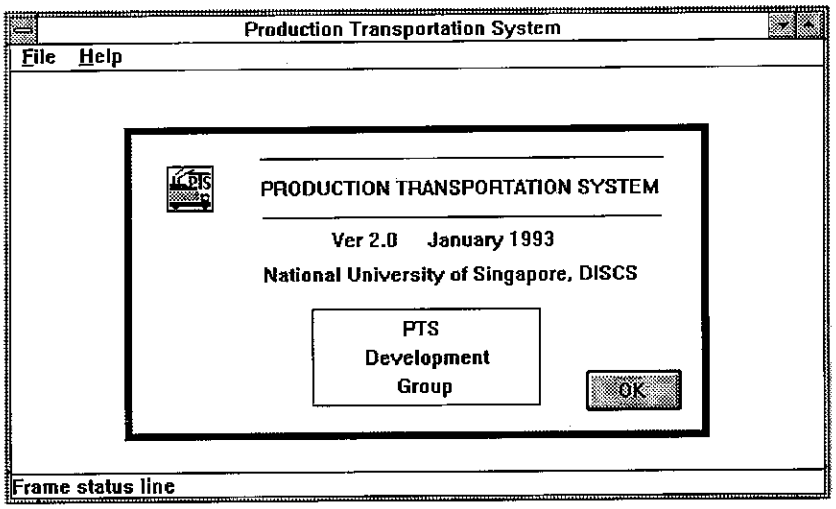

Figure 2

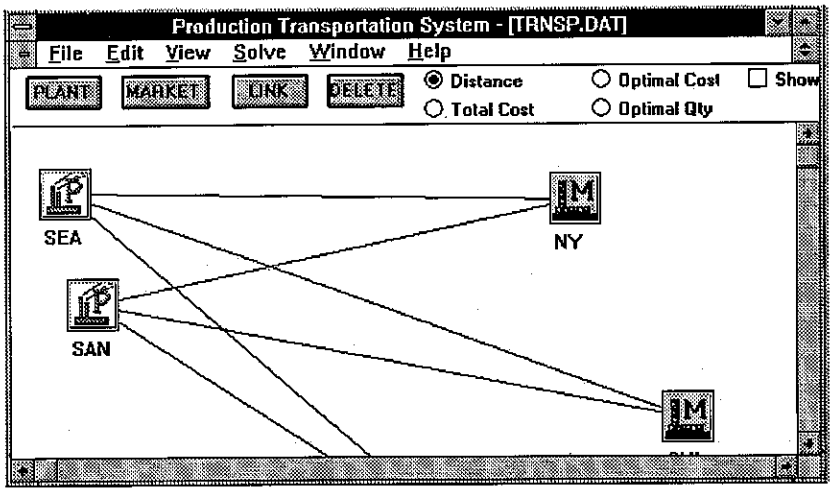

Figure 3

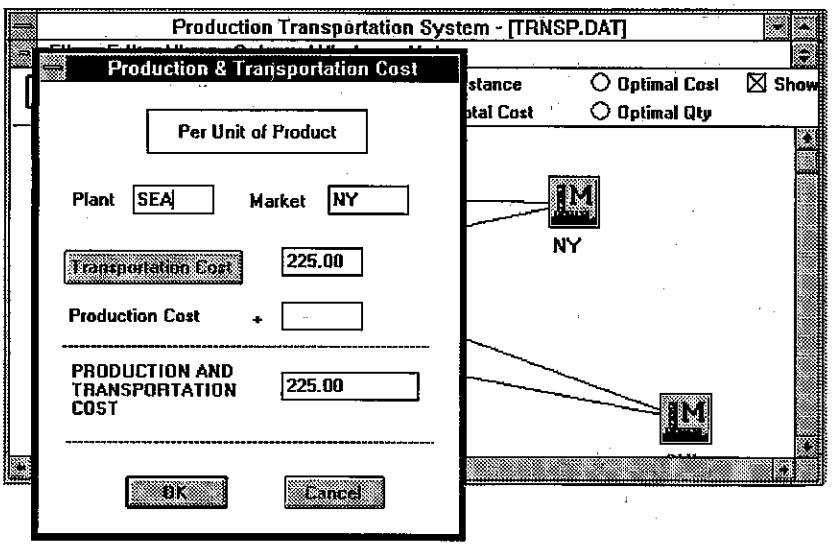

Figure 4

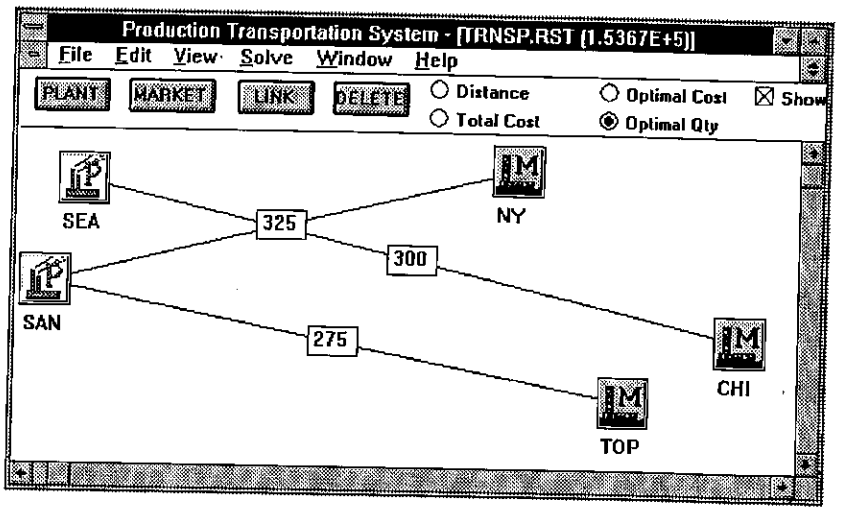

Figure 5

3. Objects in Production-Transportation Models

A simple structure diagram containing the major conceptual objects in production-transportation models is shown in Figure 6. The notations follow that of [Shlaer, 88].

A system that supports parallel modeling should contain internal structures of these objects. They will have different interface attributes in different parallel models. In the graphical model representation, for example, a PLANT object will have a PLANT symbol. In the solver representation such as GAMS, it will have reference to a specific entry in the SET definition of plants as shown underlined in Listing 1. Through message passing, functions incorporated in these different PLANT objects can be activated in parallel to achieve model consistency when data windows such as the databox shown in Figure 4 have been used for value entry.

Model-solver integration also means that result from the solver should be interpreted meaningfully in the model representation. By using solver representation templates, compilation errors from the solver can be reduced to a minimum. In our revised version of PTS, optimal solution can be interpreted correctly on the graph but there is no support in the interpretation of infeasible solution. Since these results from the solver are usually dependent on the underlying solver algorithm for solving the structured transportation problem, it is also essential to incorporate representation of the network structure.

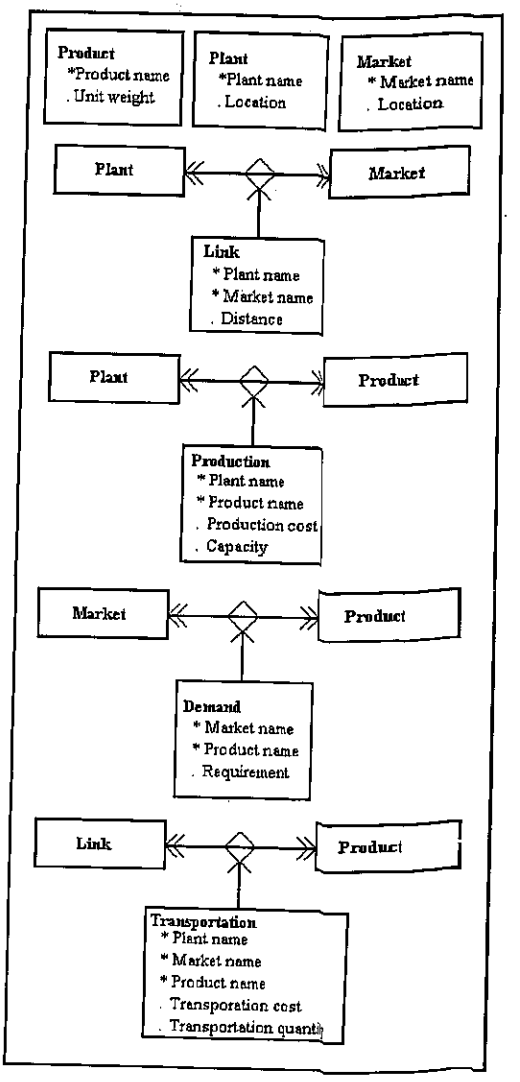

Figure 6

```
$TITLE  A TRANSPORTATION PROBLEM (TRNSPORT,SEQ=1)
$OFFUPPER
  SETS
       I   plants              / P1, P2 /
       J   markets             / M1, M2, M3 /
       K   products            / X1, X2, X3 / ;
  PARAMETER
       W(K)    unit weight of product k
           / X1   0.20
             X2   0.25
             X3   0.30 / ;
  TABLE A(I,K)  production capacities
                X1     X2     X3
         P1     40     50     60
         P2     90     80     55  ;
  TABLE B(J,K)  market demand
                X1     X2     X3
         M1     28     60     55
         M2     56     54     33
         M3     40     25     30  ;
  TABLE D(I,J)  distance between plant and market
                M1              M2              M3
      P1        2.5             1.7             1.8
      P2        2.5             1.8             1.4  ;

  SCALAR F  freight in dollars per unit weight per unit
distance  /90/ ;
  PARAMETER C(I,J)  transport cost per unit weight;
            C(I,J) = F * D(I,J) ;
  VARIABLES
       Y(I,J,K)  shipment quantities in units
       Z         total transportation costs ;
  POSITIVE VARIABLE Y ;
  EQUATIONS
       COST           define objective function
       SUPPLY(I,K)    observe supply limit at plant i
       DEMAND(J,K)    satisfy demand at market j ;
  COST ..         Z  =E=  SUM((I,J,K),
C(I,J)*Y(I,J,K)*W(K)) ;
  SUPPLY(I,K) ..   SUM(J, Y(I,J,K))  =L=  A(I,K) ;
  DEMAND(J,K) ..   SUM(I, Y(I,J,K))  =G=  B(J,K) ;
  MODEL TRANSPORT /ALL/ ;
  SOLVE TRANSPORT USING LP MINIMIZING Z ;
  DISPLAY Y.L, Y.M ;
```

Listing 1

The following example in C++ illustrates how concepts such as class inheritance and function overloading in object-oriented programming contribute to the extensibility and reusability of class objects in a production and transportation system application. Assuming that basic classes such as PLANT, MARKET and PRODUCT are defined and already instantiated in the modeler's representation, Listing 2 contains firstly a definition of an abstract class called SOURCE. Virtual functions to obtain the name and the capacity of the source for a flow in the transportation network have been declared. Two concrete classes are derived from SOURCE, PLANT_AS_SOURCE and PLANT_PRODUCT_AS_SOURCE. Each of these derived classes contain member functions of the same names, setSourceID and setCapacity. With a similar abstraction of SINK, a class representing a flow in the transportation network can now be defined as shown in Listing 3.

A transportation network can then be represented as an array of pointers to some FLOW objects, as follows:

```
class TRANSPORT_NETWORK {
  FLOW *ptflow [MAXSIZE];
  public:
    void putOptimalCost();};
```

During the modeling, either objects of PLANT_AS_SOURCE or PLANT_PRODUCT_AS_SOURCE will be created and ptflow can then be instantiated to point ultimately to these created objects so that appropriate functions applicable to the objects can be called. Should the application require objects such as machines or processes that are essential in modeling production, similar flow classes may be derived from FLOW with the necessary functions redefined.

```
class SOURCE {
  protected:
    int Cap;
    virtual char *setSourceID() = 0;
  public:
    char *ID() { return setSourceID(); };
    int Capacity() { return Cap; }; };

class PLANT_AS_SOURCE : public SOURCE {  PLANT Pl;
  virtual char *setSourceID();
  virtual void setCapacity(PLANT_CAPACITY_LIST *); };

char *PLANT_AS_SOURCE :: setSourceID() { return Pl.name(); };

void PLANT_AS_SOURCE :: setCapacity(
     PLANT_CAPACITY_LIST *pcaplist ) {
          Cap = pcaplist->retCapacity( &Pl ); }

class PLANTPRODUCT_AS_SOURCE : public SOURCE {  PLANT Pl;  PRODUCT Pr;
  virtual char *setSourceID();
  virtual void setCapacity(PLANTPRODUCT_CAPACITY_LIST *); };

char *PLANTPRODUCT_AS_SOURCE::setSourceID () {  char *source;
  source= new char[strlen(Pl.name()) + strlen(Pr.name()) + 1];
  strcpy( source, Pl.name() );
  strcat( source, Pr.name() );
  return source; }

void PLANTPRODUCT_AS_SOURCE::setCapacity(PLANTPRODUCT_CAPACITY_LIST
*ppcaplist) {       Cap = ppcaplist->retCapacity( &Pl, &Pr ); }
```

Listing 2

```
class FLOW {
  friend void setAllocation();
  friend void setCost();
  SOURCE *source;
  SINK   *sink;
  int    Alloc;
  double Cst;
 public:
   FLOW(SOURCE *so, SINK *si) { source = so; sink = si; Alloc = 0; Cst = 0
};
   char *SourceID() { return source->ID(); };
   char *SinkID() { return sink->ID(); };
   int Allocation() { return Alloc; };
   double Cost() { return Cst; }; };
```

Listing 3

4. Future works

An advantage of implementing parallel modeling on the Windows platform is the existing extensibility of employing Dynamic Data Exchange (DDE) [Petzold, 90, Chapter 17] to link the system up with other Windows applications. A standard DDE link is already provided between two Windows applications as using the Edit menu to copy and paste via the intermediate clipboard. A second method uses messaging to originate, control and end the DDE link. Two Windows applications carry on a DDE "conversation" by posting messages to each other. The programs manipulating these messages are known as the "server" and the "client." A DDE server is the program that has access to data and a DDE client is the program that may obtain the data. In the Windows environment, PTS can serve in DDE transactions both as a client to access data sources from other applications such as spreadsheets or databases, and as a server to provide result from the solution of the transportation model as data to other applications.

Future extensibility of running PTS in the Windows environment can be seen along the developement plans of Microsoft in the Windows NT™(New Technology). Among them, Object Linking and Embedding (OLE) will allow any Windows application to be executed under any other Windows application. With that possibility, model representations need

not be limited to those provided in PTS. For example, a different solver language other than GAMS can be used to provide solution to the transportation model, with data input from, and result returned to, PTS. Also, any spreadsheet or calculating tools can be opened up to help with the derivation of, say, the unit transportation cost for PTS.

Ideally, the process of PTS application development design will be a process of redesign, where the system components are iteratively refined. Besides the new derivation of FLOW classes for modeler representation as in Section 2, there may be new classes for solver representation. Thus, one may have in addition to SOLVER_TRANSPORT_MODEL, also a SOLVER_DISTANCE_MODEL to model the distance between plant and market locations, a SOLVER_PRODUCTION_MODEL to model the production details in each plant. Thus, more basic object will need to be identified that can be reused and built-upon in later extension. In effect, class libraries will be offered to developers. Thus, future work is required into studies of how these class libraries can be better built, as similarly done in [Glassey, 89].

When class libraries for general PTS have been built, the next problem is to come up with a generator using these libraries to build different user interfaces for more specific production and transportation systems. Of course, an ambitious objective would be to allow interactive specification of entire applications from these reusable objects by the end-users themselves. In other words, extensibility and reusability of object-oriented designed objects will be moved from developers to end-users. This is similar to the concepts of "visual modeling" [Angehrn, 90] or "direct manipulation" [Jones, 92] discussed in many recent articles in decision support systems. The ultimate interface will not be unlike a special Windows environment, where system-defined global objects coexist with application-defined local objects [Tello, 91]. As an illustration, consider the data windows in our prototype. PRODUCTION may be given as a global data object with product details as attributes that may or not assume values. Processes engaged for the manufacturing of these products may be important in a particular PTS application and local objects representing these processes can then be defined and integrated with the PRODUCTION global object. To this end, much can be learnt from advances made in object-oriented database systems [Kim, 90] where dynamic database evolution is an important issue.

Acknowledgements

I am grateful to Professor David Kendrick to have given me free access to materials he has collected or done before in the area of parallel model representations. The improved PTS in MDI was implemented by third year DISCS students S.H.Chang, M.H.Tay and W.B.Teng. This work was supported by NUS research grant RP609/86.

References

[Angehrn, 90]
Angehrn, A.A. and Luthi, H., Intelligent Decision Support Systems: A Visual Interactive Approach, *Interfaces*, 20:6 November-December, 1990, pp. 17-28.

[Brooke, 88]
Brooke, A., Kendrick, D., & Meeraus, A., GAMS, A User's Guide, *Scientific Press*, Redwood City, California, 1988.

[Glassey, 89]
Glassey, C.R. and Adiga, S., Conceptual design of a software object library for simulation of semiconductor manufacturing systems, *Journal of Objected-Oriented Programming*, November/December, 1989, pp. 39-43.

[Jones, 92]
Jones, C.V., "User Interface Development and Decision Support Systems", in Proceedings of the NATO Conference on Recent Developments in Decision Support Systems, Springer-Verlag, 1992.

[Kendrick, 90]
Kendrick, D.A., Parallel model representations, *Expert Systems with Applications*, Vol. 1, 1990, pp. 383-389.

[Kim, 90]
Kim, W., Introduction to object-oriented databases, *MIT Press*, 1990.

[Petzold, 90]
Petzold, C., Programming in Windows, 2nd edition, Microsoft Press, 1990.

[Shlaer, 88]
Schlaer, S. and Mellor, S.J., Object-oriented Systems Analysis, Modeling the World in Data, *Yourdon Press*, 1988.

[Tello, 91]
Object-oriented programming for Windows, *John Wiley & Sons*, 1991.