APPENDIX A: OTHER STRUCTURED METHODOLOGIES

A.1 Information Engineering Analysis (IE)

The Information Engineering Analysis (IE) method was introduced in the late 1970s when data modeling became the essential part of systems development. The history of data-oriented methodologies dates back to the 1970s when entity-relationship modeling and the relational database model were introduced. Data oriented methods make two main assumptions:

1. Well structured data establishes a firmer foundation for a system design than well structured procedures.

2. Data should be seen as an organizational resource which does not depend on data processing systems.

Data-oriented methods expanded due to the needs of a data administration system for processing organizational data.
IE is well developed and comprehensive data-oriented methods that covers the whole development cycle. Unlike structured methods, which evolved from programming, IE evolved from analysis and planning. IE provides a wider range of modeling tools and analysis techniques than structured methods: it recommends critical-success-factors analysis, enterprise modeling, data modeling, process modeling, time-box methodology, joint-requirements planning, joint-application design, and prototyping (Fichman and Keremer 1992).

There are four phases in IE:

(1) Information strategy planning
(2) Business area analysis
(3) System design
(4) Construction

These four phases are grouped into two categories of activities:

(1) Business specific (Information strategy planning and business area analysis)
(2) Project specific (System design and construction)
Figure A.1 provides a summary of tools for Information Engineering.

**Tools for Martin information engineering**

**Action diagram** – Used to depict detailed procedural logic at a given level of detail (for example, at a system level or within individual modules). Similar to structured English, except graphical constructs are used to highlight various control structures (condition, sequence, iteration, and selection).

**Bubble chart** – A low-level diagram used as an aide to normalisation of relational tables. Shows attributes (depicted as bubbles) and the functional dependencies between them (depicted as directed arcs).

**Dataflow diagram (DFD)** – Conforms to the conventional notation and usage for dataflow diagrams.

**Data-model diagram** – Depicts data entities (boxes) and their relational connections (lines). Shows cardinality and whether the connections are optional or mandatory. Similar to the entity-relationship diagram.

**Data-structure diagram** – Shows data structures in a format appropriate to the data base management system to be used for implementation.

**Encyclopedia** – A more comprehensive version of the data dictionary that serves as an integrated repository for modeling information from all development phases, including the enterprise model, organizational goals, critical success factors, strategies, and rules; data models and data definitions; process models and process definitions; and other design-related information. Automated support is assumed.

**Enterprise model** – A model that defines, at a high level, the functional areas of an organization and the relationships between them. It consists of text descriptions of functions (usually an identifiable business unit such as a department) and processes (a repetitive, well-defined set of tasks that support a function).

**Entity-process matrix** – Cross-references entities to the processes that use them.

**Process-decomposition diagram** – A hierarchical chart that shows the breakdown of processes into progressively increasing detail. Similar to the conventional tree diagram, except a particularly compact notation is used to fit many levels on one page.

**Process-dependency diagram** – A diagram consisting of processes (depicted by bubbles) and labelled arcs. It shows how each process depends on the prior execution of other processes. Similar to a dataflow diagram, except conditional logic and flow of control is also depicted.

**State-transition diagram** – Conforms to the conventional notation and usage for state-transition diagrams.

*Figure A.1: Tools for Information Engineering  
(Adapted from: Fichman and Kemerer 1992)*
During the planning stage, a high-level data architecture and an enterprise model are developed. A detailed description of business activities and their interdependencies are produced during business area analysis using tools such as decomposition diagrams, data-model diagrams, entity-process matrices, and process-dependency diagrams. A detailed model of a target system is built during the design phase using process-dependency diagrams, process-decomposition diagrams, action diagrams, dataflow diagrams, and data-structure diagrams. During the system construction phase, the models developed during the design phase are translated into an operational system, preferably using a code generator (Fichman and Kemerer 1992).

IE uses action diagrams and process-dependency diagrams for process specification. Process specification, which is separated from data specification, therefore, discourages Object-Oriented specification. Cross-referencing between functions and entities is allowed and association between event-creating operations and entities is by state-transition diagramming, hence providing a partial Object-Oriented specification. IE support type-instance concepts and encourages conceptual modeling of processes; therefore, supporting Object-Oriented concepts. Because the emphasis is placed on functional decomposition and the separation of data specification from process specification, IE cannot be considered as truly Object-Oriented, even though the static part of systems is specified by a data model comprised of entities and their relationships (Sutcliffe 1991).
A.2 Structured Systems Analysis and Design Method (SSADM)

The Structured Systems Analysis and Design Method (SSADM) is a method derived from a combination of data analysis and Structured Analysis/Structured Design (SA/SD). Dataflow diagrams are used for process analysis while entity-relationship diagrams are used for data analysis. The complete specification is functionally oriented due to the use of top-down decomposition and the separation of data from process specification. There are relationships between processing events and data objects in entity life histories, but they are merely provide one way of viewing the model within the method. Version 4 of SSADM includes entity life history as an important part of the overall specification, thereby encouraging Object-Oriented specifications. Even though data abstraction is encouraged by conceptual modeling, SSADM cannot be classified as a truly Object-Oriented method because functional modeling is supported (Sutcliffe 1990).

A.3 Structured Analysis and Design Technique (SADT)

Top-down decomposition is used in Structured Analysis Technique (SADT) to decompose systems into lower levels. Network diagrams of processes linked by dataflow, mechanisms, and control messages are used in the specification. Modeling of real-world problems is encouraged in this method, but the same arrow and box notation are used for constructs that separate data and activity models. This method places more emphasis on activity modeling. Even though it is possible to have some classification in the decomposition of data, type-instance concepts are not supported. Because data specification is separated from process specification, this method cannot be considered as an Object-Oriented approach (Sutcliffe 1990).
A.4 Information Systems Activity and Change Analysis (ISAC)

The Information Systems Activity and Change Analysis (ISAC) method uses activity diagrams for top-down decomposition of processes. In a separate specification, decomposition of data is done using data diagrams. The main emphasis is on analysis of change which supports a partial Object-Oriented approach. Processes are used for transforming data. This method does not support type-instance and classification concepts. ISAC is functionally oriented due to the separation of system from data control (Sutcliffe 1990).

A.5 Jackson System Development (JSD)

In Jackson System Development (JSD), networks of communicating processes are used for producing system models. JSD does support type-instance concepts, even though it does not support property inheritance and classification. JSD uses time-ordering of actions and their relationships with entities to model system control and more emphasis is placed on data analysis in recent versions has resulted in an object model. JSD has similar concepts to Object-Orientation, emphasizing entity-life-history approach. JSD views objects as entity roles even though it does not support object classification (Sutcliffe 1990).
A.6 Nijssen's Information Analysis Method (NIAM)

During the early stage of the analysis, Nijssen's Information Analysis Method (NIAM) concentrates on data specification. Support of data abstraction by conceptual modeling encourages Object-Orientation. Process analysis is performed by using a rule-based approach to specify data input and output transactions and by adding to the data model, semantic constraints. Even though NIAM does not support inheritance, it can be said to have some Object-Oriented properties as it supports type-instance concepts and classification by entity subtypes. However, NIAM diverges from the Object-Oriented approach because, in the data model, it emphasizes constraint-based processing firmly coupled to relationship roles (Sutcliffe 1990).

A.7 Mascot-3

Mascot promotes functional decomposition of systems. In recent versions, interfaces for system components are clearly defined and modular concepts of encapsulation are introduced. A network of communicating processes is used for system specifications, and hierarchical abstraction is also supported. Mascot supports a type-instance concept for creating many software modules instances from one template 'type'. Even though the access interface allows inheritance of procedures between modules, classification of objects is not explicitly supported. The interface specification of modules is strongly typed; therefore, encapsulation is encouraged. During early analysis, little guidance is given. Instead, other conventional methods such as CORE and structured analysis are recommended. Even though its implementation does contain Object-Oriented features, Mascot encourages a functionally oriented specification because functional decomposition is emphasized and structured methods are used during its early stages (Sutcliffe 1990).
APPENDIX B: OTHER OBJECT-ORIENTED METHODOLOGIES

B.1 Booch Object-Oriented Design (OOD)

Booch Object-Oriented Design (OOD) process contains four steps (Booch 1991):

1. **Identifying classes and objects.** Important classes and objects are identified and mechanisms are invented. ‘A mechanism is a structure whereby a set of objects work together to provide a behavior that satisfies some requirement of the problem’ (Booch 1991:123).

2. **Identifying the semantics of classes and objects.** The meanings of the objects and classes from the previous step are established. Different techniques are used, including defining each object’s life cycle from creation to destruction using scripts.

3. **Identifying relationships among classes and objects.** Firstly, class and object interactions are established using relationships such as inheritance. Secondly, decisions about the visibility between objects and classes must be made.

4. **Implementing classes and objects.** The inside of each object or class is looked at, and decisions made regarding how its behavior (services) should be implemented. Also, objects and classes are allocated to modules and programs are allocated to processors.
### Tools for Booch Object-Oriented Design

**Action diagram** – Shows the existence of classes (enclosed in dotted-line "clouds") and their relationships (depicted by various kinds of directed and undirected arcs) in the logical design of a system. Relationships supported include uses, instantiates, inherits, metaclass, and undefined.

**Module diagram/template** – Documents the allocation of objects and classes to modules in the physical design of a system. Only need for languages (such as Ada) that support the idea of a module as distinct from objects and classes.

**Object diagram/template** – Used to model some of dynamics of objects. Each project (enclosed in solid line "clouds") represents an arbitrary instance of a class. Objects are connected by directed arcs that define object visibility and message connections. Does not show flow of control or ordering of events.

**Operation template** – Structured text that provides detailed design documentation for operations.

**Process diagram/template** – Used to show the allocation of processes to processors in the physical design of a system. Only for implementations in multiprocessors environments.

**State-transition diagram** – Shows the state (depicted by circles) of a class, the events (directed arcs) that cause transitions from one state to another, and the actions that result from a state change.

**Timing diagram** – A companion diagram to the object diagram, shows the flow of control and ordering of events among a group of collaborating objects.

---

*Figure B.1: Tools for Booch Object-Oriented Design*  
*(Source: Fichman and Kemerer 1992)*
B.2 Object-Oriented Modeling and Design (OMT)

Object-Oriented Modeling and Design (OMT) consists of three main phases: analysis, system design, and object design (Rumbaugh et al. 1991).

During the analysis stage, a system is described using three models:

1. The object model which describes the objects in a system and their relationships with one another. Object diagrams are used. The object model is constructed using the following steps:
   - Identifying objects and classes
   - Prepare a data dictionary
   - Identifying associations (including aggregations) between objects
   - Identifying attributes of objects and links
   - Organize and simplify object classes using inheritance
   - Verify that access paths exist for likely queries
   - Iterate and refine the model
   - Group classes into modules
   (Rumbaugh et al. 1991:152)

2. The dynamic model which describes the interactions between the objects in the system. State diagrams are used. The dynamic model is constructed using the following steps:
   - Prepare scenarios of typical interaction sequences
   - Identify events between objects
   - Prepare an event trace for each scenario
   - Build a state diagram
   - Match events between objects to verify consistency
   (Rumbaugh et al. 1991:170)
3. The functional model which describes the transformations of data within a system. Data flow diagrams are used. The functional model is constructed using the following steps:

- Identify input and output values
- Build data flow diagrams showing functional dependencies
- Describe functions
- Identify constraints
- Specify optimization criteria

(Rumbaugh et al. 1991:180)

During the system design stage, the following decisions must be made by the system designer:

- Organize the system into subsystems
- Identify concurrency inherent in the problem
- Allocate subsystems to processors and tasks
- Choose an approach for management of data stores
- Handle access to global resources
- Choose the implementation of control in software
- Handle boundary conditions
- Set trade-off priorities

(Rumbaugh et al. 1991:199)
During the object design stage, the following steps must be performed by the designer:

- Combine the three models to obtain operations on classes
- Design algorithms to implement operations
- Optimize access path to data
- Implement control for external interactions
- Adjust class structure to increase inheritance
- Design associations
- Determine object representation
- Package classes and association into modules

Rumbaugh et al. (1991:228)

Refer to Rumbaugh et al. (1991) for notations and further discussion.

B.3 Wirfs-Brock et al. Responsibility-Driven Design (RRD)

Responsibility driven design (RRD) uses the client-server model approach in which systems consist of clients and servers (Wirfs-Brock, Wilkerson, and Wiener 1990). Servers have their own private responsibilities as well as providing services to clients by contracts.

The following six steps are performed by RRD:

(1) Identifying Classes. A list of classes are built by finding nouns that describe conceptual entities, physical objects, external interfaces, and categories of objects in the requirements specification. Attributes of super-classes and objects are also identified.
(2) Identifying responsibilities (operations) and assigning to classes. The purpose of each class is considered and responsibilities is derived from action phrases in the specification. Responsibilities are then assigned to classes.

(3) Finding collaborations (object interactions). Responsibilities related to each class are examined to see which other classes are required for collaboration, so that each responsibility can be fulfilled.

(4) Defining hierarchies. Class hierarchies are constructed for inheritance relationships. Responsibilities employed by the same clients are grouped together to form contracts.

(5) Defining subsystems. A collaborations graph is drawn for the whole system. Frequent and complex collaboration are searched for and defined as subsystems.

(6) Defining protocols (specification of object interfaces). Design detail is developed by writing specifications for contracts, subsystems, and classes. The protocols for each class are constructed.
Tools used for RDD is shown in Figure B.2.

<table>
<thead>
<tr>
<th>Tools for Wirfs-Brock et al. Responsibility-Driven Design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class cards</strong> – A physical card used to records text describing classes, including name, superclasses, subclasses, responsibilities, and collaborations.</td>
</tr>
<tr>
<td><strong>Class specification</strong> – An expanded version of the class card. Identifies superclasses, subclasses, hierarchy graphs, collaborations graphs. Also includes a general description of the class, and documents all of its contracts and methods.</td>
</tr>
<tr>
<td><strong>Collaborations graph</strong> – A diagram showing the classes, subsystems, and contracts within a system and the path of collaboration between them. Classes are drawn as boxes. Subsystems are drawn as rounded-corner boxes enclosing multiple classes. Collaborations are directed arcs from one class to the control of another class.</td>
</tr>
<tr>
<td><strong>Hierarchy diagram</strong> – A simple diagram that shows inheritance relationships in a lattice-like structure. Classes (enclosed within boxes) are connected by undirected arcs that represent an inheritance relationship. Superclasses appear above subclasses.</td>
</tr>
<tr>
<td><strong>Subsystem card</strong> – A physical card used to record text describing subsystems, including name and a list of contracts.</td>
</tr>
<tr>
<td><strong>Subsystem specification</strong> – Contains the same information as a class specification, only at the level of a subsystem.</td>
</tr>
<tr>
<td><strong>Venn diagram</strong> – Used to show the overlap of responsibilities between classes to help identify opportunities to create abstract superclasses. Classes as depicted as intersecting ellipses.</td>
</tr>
</tbody>
</table>

*Figure B.2: Tools for Wirfs-Brock et al. Responsibility-Driven Design
Source: Fichman and Kemerer 1992*
B.4 Shlaer and Mellor Object-Oriented Systems Analysis (OOSA)

Shlaer and Mellor Object-Oriented Systems Analysis (Shlaer and Mellor 1988 and 1992) uses three models to represent the system to be developed: an information model, a state model, and a process model. The following six-step procedure is employed by Shlaer and Mellor:

1) *Developing and information model.* The objects, their attributes, and relationships between them are depicted on this model. Objects used here are basically the same as entities in the structured methods.

2) *Defining object life cycles.* The state model is constructed. This is composed of a collection of states (each stage in the life cycle of an object is represented by a state), events (an event causes an object to change its states), transition rules (a transition rule specifies the new state of an object when it receives a certain event), and actions (an action is an operation that must be performed by an object when a new state is attained). A timer is also defined. ‘A timer is a mechanism that can be used by an action to generate an event at some time in the future’ (Shlaer and Mellor 1992:52).

3) *Defining the dynamics of relationships.* A state model is developed for dynamic relationships between objects. A dynamic relationship is a relationship that changes over time. An association object is defined in the information model for each dynamic relationship. For relationships where there may be competition between objects for resources of another object, special assigner state models are constructed.
(4) *Defining system dynamics.* At this point a model of control and time is produced at the system level. Asynchronous control (similar to message passing) is showing using an object-communication model (OCM). Synchronous control (when the instance data of an object is accessed by another through an accessor process) is shown using an object-access model. A procedure for keeping tracks of threads of control at a high level (using events on the OCM) and at a lower level (using a thread of control chart) is also described.

(5) *Developing process models.* A dataflow diagram is created for each action that shows all of the processes for that action, and the data flows between the processes and data stores. There are four types of processes: accessors, event generators, transformations, and tests. Guidelines are also provided for separating actions into these constituent processes.

(6) *Defining domains and subsystems.* The subject matter can be decomposed into conceptually distinct domains. There are four types of domains: application, services, architectural, and implementation. The application domain can be decomposed into multiple subsystems.
Tools for OOSA is shown in Figure B.3.

**Tools for Shlaer and Mellor Object-Oriented Systems Analysis**

**Action-dataflow diagram (ADFD)** - Similar to DFDs, except ADFDs are used to model elementary "action" processes rather than to create a top-down functional decomposition of the entire system. Standard DeMarco notation is used, except additional notations are provided to show control flows and to show conditionality in the execution of dataflows and control flows.

**Domain chart** - A simple diagram that illustrated all domains relevant to the implementation of an OOA model. Domains are enclosed within bubbles and are connected by directed arcs. These arcs represent bridges between domains. Four types of domains are identified: application, service, architectural, and implementation.

**Information structure diagram** - A variant on the entity-relationship diagram that shows objects (boxes) connected by relationships (labelled arcs). Attributes are listed within object boxes. Relationship conditionality and multiplicity are also shown.

**Object and attribute description** - A text description of an object, including object name, object description, object identifier, a list of attributes, and description of each attribute.

**Object-access model** - Shows the synchronous interactions between state models at the global system level. Synchronous interactions occur when one state model accesses the instance data of another object via an accessor process. State models (enclosed in ovals) are connected to each other by directed arcs labelled with the accessor process.

**Object-communication model** - Shows the asynchronous interactions between state models and external entities at the global system level. State model (enclosed in ovals) are connected to each other and to external entities (enclosed in boxes) by directed arcs labelled with communicating events.

**Process description** - A narrative description of a process. A process description is needed for every process appearing on an action-dataflow diagram.

**Relationship specification** - A text description of each relationship, including the name of the relationship (from the point of view of each object), conditionally (required or optional), multiplicity (one-to-one, one-to-many, many-to-many), a general description of the relationship, and identification of the attributes (foreign keys) through which the relationship is formalised.

**State model** - State models conforms to the conventional notation for state-transition diagrams, except they are used to model the states of problem domain entities. (Traditionally STDs, by contrast, model the states of a system, system component, or process.)

**Subsystem access model** - Shows synchronous interactions between object-access models (one OAM exists for each subsystem). Directed, labelled arcs represent synchronous processes flowing between OAMs (enclosed in boxes).

**Subsystem communication model** - Shows asynchronous interactions between object-communication models (one OCM exists for each subsystem). Directed, labelled arcs represent asynchronous events flowing between OAMs (enclosed in boxes).

**Subsystem relationship model** - Shows relationships between information models (where each subsystem has exactly one information model). Information models (enclosed in boxes) are connected by undirected arcs (labelled with relationships).

*Figure B.3: Tools for Shlaer and Mellor Object-Oriented Systems Analysis (Source: Fichman and Kemerer 1992)*
B.5 Bailin Object-Oriented Requirements Specification Method (OOS)

There are two distinctions which are central to Bailin Object-Oriented Requirements Specification Methods (OOS) Bailin 1989): a distinction between entities which have states and functions which do not have states. Entities can be decomposed further into sub-entities and functions, but functions can only be decomposed into subfunctions; a distinction between active entities which need to be considered in dept during the analysis phase because of the important operations they perform, and passive entities which can be left out until the design phase. These distinctions are important because functions, active entities, and passive entities are modeled differently during the analysis phase.

The OOS method uses even-step procedure:

1. **Identifying key problem domain entities.** Dataflow diagrams are drawn. Objects that appear as candidate entities in process names are then assigned.

2. **Distinguishing between active and passive entities.** Entities whose operations are important for describing system requirements (active entities) are distinguished from those whose operations can be postponed until design (passive entities). An Entity-Relationship diagram (ERD) is constructed.

3. **Establishing dataflows between active entities.** A Top-level entity-dataflow diagram (EDFD) is constructed. Each active entity is assigned as a process node. Each passive entity is specified as a data store or dataflow.

4. **Decomposing entities (or functions) into subentities and/or functions.** Step 5 and 6 are performed iteratively together with this step. Each active entity in the top-level EDFD is considered to determine whether it contains lower level entities. The operations performed by each entity are also considered and assigned as functions. A new EDFD is constructed for each of the subentities found. This process of decomposition is to be continued.
(5) **Checking for new entities.** New entities are checked at each stage of decomposition and added them to the appropriate EFFD.

(6) **Grouping functions under new entities.** Functions performed by new entities are identified. If necessary EDFDs are reorganized and changed from passive to active entities.

(7) **Assigning entities to appropriate domains.** Each entity is to be assigned to some application domain, and a set of ERDs to be created, one for each domain.

The tools used for OOS are shown in Figure B.4.

<table>
<thead>
<tr>
<th>Tools for Bailin Object-Oriented Requirements Specification Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity-relationship diagram</strong> – Conforms to the conventional notation and usage for entity-relationship diagrams.</td>
</tr>
</tbody>
</table>

**Entity-dataflow diagram (EDFD)** – A variant on the conventional dataflow diagram wherein each process node contains either an active entity or some function relation to an active entity, rather than disembodied processes. Active entities and functions are enclosed within bubbles. Bubbles are connected to each other and to data stores by labelled arcs containing data-flows. Dataflows and data stores are passive entities.

**Entity dictionary** – A repository of entity names and description, analogous to the data dictionary of DeMarco structured analysis.

*Figure B.4: Tools for Bailin Object-Oriented Requirements Specification Methods (Source: Fichman and Kemerer 1992)*
APPENDIX C

C.1 CASE STUDY ONE

University Course Registration System

A Student completes a registration request form and mails or delivers it to the registrar’s office. A clerk enters the request into the system. First, the Accounts Receivable subsystem is checked to ensure that no fees are owed form the previous quarter. Next, for each course, the student transcript is checked to ensure that the course prerequisites are completed. Then, class position availability is checked. If all checks are successful, the student’s identity card number is added to the class list.

The acknowledgement back to the student shows the result of registration processing as follows: If fees are owing, a bill is sent to the student; no registration is done and the acknowledgement contains the amount due. If prerequisites for a course are not filled, the acknowledgement lists prerequisites not met and that course is not registered. If the class is full, the student acknowledgements marked with “course closed”. If a student is accepted into a class, the day, time, and room are printed next to the course number. Total tuition owed is computed and printed on the acknowledgement. Student fee information is interfaced to the Accounts Receivable subsystem. Course enrollment reports are prepared for the instructors. The instructors would then be able to access the enrolment sheet via the Faculty Information System through the input of his/her passwords.
C.1.1 System Interaction Diagram (SID) for the Student Registration System.

C.1.2 Class Relationship Diagram (CRD) for the Student Registration System.
C.1.3 Object Interaction Diagram (OID) for the Student Registration System.

C.1.4 State Evolution Diagram (SED) for the Student Registration System.
C.2 CASE STUDY TWO

I Library Management System

The text below describes an example on the functioning of a library system for a university, for which the purpose of the system, the organizational choices and management rules are described separately after discussion with managers and the librarians of the library.

II Purpose

The aim of the project is to organize the functioning of a library, stressing on obtaining a prototype for an automated library information system (IS). The library IS should allow a coherent functioning of the library and inform the librarians of the present or former state of its functioning, bringing with it, for example, information on:

- the inventory of books by subject, or author, year of entry or keywords.
- The inventory of books borrowed at a given date.
- The inventory of present or former borrowers, or for a given period.
- The list of books overdue.
- The books borrowed at a given date, by a given borrower.

The library IS is also expected to provide services to the Faculty Information System (FIS) and to feed the Accounting Management System (AMS) with monthly income and expenditure information for the overall operation of the library. Each faculty member should be able to request for books as well as to be able to check the inventory of books/journals available in the library. The system should be able to provide the status of books ordered by the faculty members.
III Organizational Choices

The library has a collection of books, each with a unique reference number. There may be several copies of the same book in the library. In the actual management, every new book acquired by the library will be given a new unique reference number. The reference number and the price of the book, the names of its author(s) and editor, the date of its edition and a summary in the form of a list of its keywords, the number of its copies for loan, and the date on which its particular copy is put out for loan, will be stored in an object base. Any deletion of copies or of information concerning a copy of book will also be updated in the object base.

Among other needed information stored in the object base are the details of a borrower such as name, identification number, address, telephone number, occupation, will also be stored in the object base. The date of each loan, the books borrowed and the date of each return will also be recorded.

IV Rules of Management

Below is a list of principal rules of the library functioning.

- To register as a borrower, a registration fee must be paid and is renewable every year. The period of registration is 12 months from the date of payment of the registration.

- The maximum delay allowed for the payment of a renewed registration is a month from the normal date of renewed registration.

- The borrower who has not paid his/her registration fee a month after the normal delay cannot borrow any book. He/She is said to be suspended.

- A borrower can, at the most, borrow 3 books.
• Any book on loan must be returned to the library, at the latest, the end of the period of a month for the loan.

• The requests for loan non-satisfied are placed on a waiting list, if the borrower wishes; the request would be satisfied according to the chronological order of their requests.

• When a book on the waiting list demanded by a borrower becomes available, he/she is informed by notice. He/She has a week to come to take it. At the end of this delay, the book will no longer be reserved for him.

• A request on the waiting list for 3 months should be cancelled

• The borrower who has not returned a book borrowed within the normal allowable period will be reminded one week after the expiry of the date of return; he/she will be fined at the end of 2 weeks of delay. The fine is equivalent to 20% of the price of the book. Each additional week of delay gives rise to a new reminder and to a fine of the same amount.

• A borrower who has not returned a borrowed book to the library 2 weeks after the allowable period of a month will be suspended from borrowing any book. If he/she has been fined thrice for the same loan, he/she can no longer benefit from the services of the library. He/She is said to be excluded.

• At the moment of annual inventory, the librarian can decide to delete copies of books too worn out and proceed to request new ones. He/She can decide to purchase additional copies of books much in demand.
C.2.1 System Interaction Diagram (SID) for the Library Management System.

C.2.2 Class Relationship Diagram (CRD) for the Library Management System.
C.2.3 Object Interaction Diagram (OID) for the Library Management System.

C.2.3.1 Account Processing

C.2.3.2 Faculty Requisition
C.2.3.3 Loan Processing

C.2.4 State Evolution Diagram (SED) for the Library Management System.

C.2.4.1 Loan
C.2.4.2 Book

C.2.4.3 Borrower
C.3 CASE STUDY THREE

Dr. Patel's Dental Practice System

The dental practice uses a manual patient and billing system to serve approximately 1,100 patients. The primary components of the manual system are scheduling patient appointments, maintaining patient dental records, and recording financial information. Due to increased competitive pressure, Dr. Patel desires to automate his customer records and billing.

New patients must complete the patient history form. The data elements are listed in Table 1. Then, at the first visit, the dentist evaluates the patient and completes the second half of the patient history information with the standard dental codes (there are 2,000 codes) to record recommended treatments. The data elements completed by the dentist are listed as Table 2. The patient history form is filed in a manila folder, with the name of the patient as identification, along with any other documents from subsequent visits.

The calendar of appointments is kept by the secretary, who schedules follow-up visits before the patient leaves the office. The calendar data elements are shown as Table 3. Also, before the patient leaves, any bills, insurance forms, and amounts due are computed. The client may pay at that time, or may opt for a monthly summary bill. The secretary maintains bill, insurance, and payment information with the patient history. Financial data elements are shown in Table 4. Every week, the secretary types mailing labels that are attached to appointment reminder cards and mailed. Once per month, the secretary types and sends bills to clients with outstanding balances.
TABLE 1  Patient History Information

Patient Name, Address, City, State, PostCode, Home telephone, Date of Birth, Sex, Parent’s name (if under 21) or emergency contact (Address, City, State, PostCode, Telephone number), Known dental problems, Known physical problems, Known drug/medication allergies.

Place of work name
   Address, city State, PostCode, Telephone Number.

Insurance Carrier
   City, State, PostCode, Policy Number.

Last dentist name
   Address, City, State, PostCode.

Physician Name
   City, State, PostCode.

---

TABLE 2  Dentist Prognosis Information

Dentist performing evaluation
   Date of evaluation
   Time of evaluation
   Recommended treatment
   Procedure Code
   Date Performed (Completed when performed)
   Fee (Completed when performed)

---

TABLE 3  Appointment Calendar

Patient Name
   Home telephone number
   Work telephone number
   Date of last service
   Date of appointment
   Time of appointment
   Type of treatment planned

---

TABLE 4  Patient Financial Information

Patient Name
   Address
   City, State, PostCode
   Home telephone number
   Work telephone number
   Date of service
   Fee
   Payment received
   Date of payment
   Adjustment
   Date of Adjustment
   Outstanding Balance
   Date bill sent
   Date overdue notice sent
C.3.1 System Interaction Diagram (SID) for the Dental Management System.

C.3.2 Class Relationship Diagram (CRD) for the Dental Management System.
C.3.3 Object Interaction Diagram (OID) for the Dental Management System.

C.3.4 State Evolution Diagram (SED) for the Dental Management System.


C.4 CASE STUDY FOUR

Alarm Design System

There are several kinds of design component types in the design model of a component system:

- The internal design component types that are the basic components of the component system;
- The concrete design component types provided for reuse without adaptations;
- The abstract design component types provided to anticipate future adaptations, in each of which are integrated one or several variation points supported by one or several variability mechanisms.

The systems in the alarm domain handle a number of sensors, detectors, manual call points, actuators and presentations units:

- Sensor: the sensor is a device that detects an alarm state. It returns a multi-valued signal, e.g., 1 to 255. An alarm is actuated when the value exceeds a predefined limit.

- Detector: the detector is a device that detects an alarm state. It returns a discrete signal (alarm/not alarm). An alarm is actuated if a signal is received from the detector.

- Manual call point: this is a button which can be pressed by a person. An alarm is actuated when the button is pressed.

- Actuator: a device for producing the alarm signal.

- Presentation unit: the presentation unit presents the status information and allows the system to be controlled.
The system differ in the type of alarms they handle, their size, security level, cost, user interaction and so on. The common feature is that they react to signals from sensors and detectors, perform some action using different actuators, and display their state using a presentation unit. There is an alarm transmitter which can either be one of the three specific devices: telephone, fax and computer, for transmitting the alarm. Depending on the types of sensors and detectors and actuators used, an alarm system can be configured as a burglar alarm system, a fire alarm system, a car alarm system and so on. Alarm systems are developed for different hardware systems using different embedded software.

C.4.1 **System Interaction Diagram (SID) for the Alarm System.**
C.4.2 Class Relationship Diagram (CRD) for the Alarm System.
C.4.3 Object Interaction Diagram (OID) for the Alarm System.

C.4.3.1 Sensor

C.4.3.2 Detector
C.4.4 State Evolution Diagram (SED) for the Alarm System.
APPENDIX D

SOURCE CODES

Method Definition Files For the HYCONAN Model

(Refer to the equivalent OPPR specification in Section 5.15, Figure 5.8).

Method Definition File of the CRD Metamodel

/* Author      : Vasuthevan Balakrishnan   */
/* Project     : HYCONAN - Graphics Editor */
/* Date Completed: 15/4/1999             */
metamodel "HYCONAN Class Relationship Diagram (CRD)"
} extension "CRD"

/* SHAPE DEFINITIONS */

shape "Rectangle"
{ shape (0@0,200@0,200@200,0@200);
  connection points (0@0,200@0,200@200,0@200);
  line type solid;
  line width 1;
  color black filled white;

shape "Upperline"
{ shape (0@80,200@80);
  line type solid;
  line width 1;
}

shape "Lowerline"
{ shape (0@140,200@140);
  line type solid;
  line width 1;
}
shape "Invisible Rectangle"
  { shape (0@0.200@0,200@50,0@50);
    line type invisible;
    connection points (0@0.200@0,200@50,0@50);
  }
shape "Possession Arrow"
  { shape (90@100,80@95,80@105);
    line type solid;
    line width 0;
    color black filled black;
  }

/* SYMBOL DEFINITIONS */
symbol "Rectangle"
  { shapes ("Rectangle", "Upperline", "Lowerline");
    scale 0.5;
    labels {"Name" at (1 1 200 25) centered;
           "Characteristics" at (1 16 200 30) centered;
           "Kind" at (1 26 200 50) centered;
           "Parameters or Arguments" at (1 51 200 75) centered;
           "Attributes" at (1 81 200 140) centered;
           "Operations" at (1 141 200 200) centered;
    }
  }
symbol "Note"
  { shapes ("Invisible Rectangle");
    scale 0.5;
    labels {"Related item" at (1 1 200 25) centered;
             "Detail" at (1 26 200 50) centered;
    }
  }
symbol "Arrow"
  { shapes ("Arrow");
    scale 1.0;
  }
symbol "Arrow 2"
  { shapes ("Arrow");
    scale 1.0;
    labels {"Role name" at (45 105 90 130) centered;
            "Cardinality" at (45 70 90 95) centered;
            "Association name" at (0 70 35 95) centered;
    }
symbol "Arrow 1"
{ shapes ("Arrow");
  scale 1.0;
  labels {
    "Cardinality" at (45 105 90 130) centered;
    "Role name" at (45 70 90 95) centered;
  }
}

symbol "Generic class"
{ shapes ("Possession Arrow", "Arrow");
  scale 1.0;
}

symbol "Specific class"
{
  scale 1.0;
  labels {"Kind of possession" at (45 70 90 95) centered;
          "Cardinality" at (45 105 90 130) centered;}
}

/* PROPERTY DEFINITIONS */

property type "Name"
{ datatype String;
  values unique;
}

property type "Kind"
{ datatype editlist("{Abstract}"," ");
}

property type "Parameters or Arguments"
{ datatype String;
}

property type "Characteristics"
{ datatype String;
}
property type "Attributes"
{ datatype Text;

}

property type "Operations"
{ datatype Text;

}

property type "Cardinality"
{ datatype editlist ("1", "+1", ",", ",..");

}

property type "Role name"
{ datatype String;

}

property type "Association name"
{ datatype editlist ("-->", "<--");

}

property type "Kind of possession"
{ datatype editlist ("{V}", ", {R}", ",");

}

property type "Related item"
{ datatype String;
values unique;

}

property type "Detail"
{ datatype Text;

}
/* OBJECT DEFINITIONS */

object type "Note"
{ symbol "Note";
duplicates allowed;
properties ("Related item", "Detail");
}

object type "Class"
{ symbol "Rectangle";
duplicates allowed;
properties ("Name", "Kind", "Parameters or Arguments", "Attributes","Operations");
}

relationship type "Association"
{
  line type solid;
directed;
line width 0;
}

relationship type "Inheritance"
{
  line type solid;
directed;
line width 0;
}

relationship type "Possession"
{
  line type solid;
directed;
line width 0;
}

relationship type "Instantiation"
{
  line type dot;
directed;
line width 0;
}

role type "First associated class" of "Association"
{ symbol "Arrow 1";
  connectivity (0,M);
  properties ("Cardinality", "Role name");
}

role type "Second associated class" of "Association"
{
symbol "Arrow 2";
connectivity (0,M);
properties ("Association name","Cardinality", "Role name");
}

role type "Specific class" of "Possession"
{
    symbol "Specific class"
    connectivity (0,M);
}

role type "Generic class" of "Possession"
{
    symbol "Generic class"
    connectivity (0,M);
}

role type "Subclass" of "Inheritance"
{
    connectivity (0,M);
}

role type "Superclass" of "Inheritance"
{
    symbol "Arrow"
    connectivity (0,M);
}

role type "Instantiated class" of "Instantiation"
{
    connectivity (0,M);
}

role type "Generic class" of "Instantiation"
{
    symbol "Arrow"
    connectivity (0,M);
}
*/ BINDING DEFINITIONS */

bind relationship "Association"
as "First associated class" ("Class"),
as "Second associated class" ("Class")
bind relationship "Possession"
as "Specific class" ("Class"),
as "Generic class" ("Class")
bind relationship "Inheritance"
as "Subclass" ("Class"),
as "Superclass" ("Class")
bind relationship "Instantiation"
as "Instantiated class" ("Class"),
as "Generic class" ("Class")

Method Definition File of the CID Metamodel

(Refer to the equivalent OPPR specification in Section 5.15, Figure 5.10).

/* Author : Vasuthevan Balakrishnan */
/* Project : HYCONAN - Graphics Editor */
/* Date Completed : 15/4/1999 */

metamodel "HYCONAN Class Interaction Diagram (CID)"
extension "CID"

/* SHAPE DEFINITIONS */

shape "Rectangle"
{ shape (0@0,200@0,200@100,0@100);
  connection points (0@0,200@0,200@100,0@100);
  line type solid;
  line width 1;
  color black filled white;
}

shape "Polygon"
{shape (0@100,50@0,150@0,200@100,150@200,50@200,0@100);
  connection points (0@100,100@0,200@100);
  line type solid;
  line width 1;
  color black filled white;
}

shape "Arrow"
{shape (100@100,90@95,90@105);
  line type solid;
"Sequence number" at (45 105 100 130) centered;
"Synchronization" at (45 105 130 150) centered;

/* PROPERTY DEFINITIONS */

property type "Class name"
{datatype String;
 values unique;
}

property type "Process name"
{ datatype String;
  values unique;
}

property type "Concurrency"
{ datatype editlist ("{Act}"));
}

property type "Related item"
{ datatype String;
  values unique;
}

property type "Detail"
{ datatype Text;
}

property type "Signature"
{ datatype String;
}

property type "Sequence number"
{ datatype editlist ("{1}"","{2}"","{3}"","{4}"","{5}"","{6}"","{7}"","{8}"","{9}"","{10}")
}

property type "Synchronization"
{ datatype editlist ("{S}"","{B}"","{T}"","{A}"));
}

property type "Visibility"
{ datatype editlist ("{P}"","{L}"));
}

property type "Constraints"
{ datatype editlist ("{ }"));
}

/* OBJECT DEFINITIONS */

object type "Note"
{symbol "Invisible Rectangle";
duplicates allowed;
properties ("Related item","Detail");
}

object type "Class"
{symbol "Rectangle";
duplicates allowed;
properties ("Class name","Concurrency");
}

object type "Process"
{symbol "Polygon";
duplicates allowed;
properties ("Process name");
}

/* RELATIONSHIP DEFINITIONS */

relationship type "Message Passing"
{line type solid;
directed;
line width 0;
}

relationship type "ProcessClass Message Passing"
{line type solid;
directed;
line width 0;
}

relationship type "ClassProcess Message Passing"
{line type solid;
directed;
line width 0;
}

/* ROLE DEFINITIONS */

role type "Client object" of "ProcessClass Message Passing"
{symbol "Line";
connectivity (0,M);}

role type "Server object" of "ProcessClass Message Passing"
{symbol "Arrow";
connectivity (0,M);
}

role type "Client object" of "ClassProcess Message Passing"
{symbol "Line";
connectivity (0,M);

role type "Server object" of "ClassProcess Message Passing"
{symbol "Arrow";
  connectivity (0,M);
}

role type "Client class" of "Message Passing"
{symbol "Line";
  connectivity (0,M);
  properties ("Visibility","Constraints");
}

role type "Server class" of "Message Passing"
{symbol "Arrow";
  connectivity (0,M);
  properties ("Signature","Sequence number","Synchronization");
}

/* BINDING DEFINITIONS */

bind relationship "Message Passing"
as "Client class" ("Class"),
as "Server class" ("Class")
bind relationship "ProcessClass Message Passing"
as "Client object" ("Process"),
as "Server object" ("Class")
bind relationship "ClassProcess Message Passing"
as "Client object" ("Class"),
as "Server object" ("Process")

Method Definition File of the SID Metamodel

(Refer to the equivalent OPPR specification in Section 5.15, Figure 5.11).

/* Author      : Vasuthevan Balakrishnan */
/* Project     : HYCONAN - Graphics Editor */
/* Date Completed : 15/4/1999 */

metamodel "HYCONAN System Interaction Diagram (SID)"
extension "SID"

/* SHAPE DEFINITIONS */

shape "Rectangle"
{ shape (0@0,200@0,200@100,0@100);
  connection points (0@0,200@0,200@100,0@100);
  line type solid;

194
scale 1.0;
lables {
"Signature" at (45 70 90 95) centered;
"Sequence number" at (45 105 90 130) centered;
"Status" at (45 105 90 130) centered;
}

/* PROPERTY DEFINITIONS */

property type "System name"
{ datatype String;
  values unique;
}

property type "Focus System name"
{ datatype String;
  values unique;
}

property type "Process name"
{ datatype String;
  values unique;
}

property type "Related item"
{ datatype String;
  values unique;
}

property type "Detail"
{ datatype Text;
}

property type "Signature"
{ datatype String;
}

property type "Sequence number"
{ datatype editlist ("{1}"", "{2}"", "{3}"", "{4}"", "{5}"", "{6}"", "{7}"", "{8}"", "{9}"", "{10}",
  "{11}"", "{12}"", "{13}"", "{14}"", "{15}"", "{16}"", "{17}"", "{18}"", "{19}"", "{20}"”);
}

property type "Status"
{ datatype editlist ("{A}"", "{R}"", "{T}"”);}

property type "Constraints"
{ datatype editlist ("{ }”);}

/* OBJECT DEFINITIONS */

object type "Note"
{ symbol "Invisible Rectangle";
  duplicates allowed;

properties ("Related item", "Detail");
}

object type "System"
{
    symbol "Rectangle";
    duplicates allowed;
    properties ("System name");
}

object type "Focus System"
{
    symbol "Solid Rectangle";
    duplicates not allowed;
    properties ("Focus System name");
}

object type "Process"
{
    symbol "Ellipse";
    duplicates allowed;
    properties ("Process name");
}

/* RELATIONSHIP DEFINITIONS */

relationship type "FocusSystem-System passing"
{
    line type solid;
    directed;
    line width 0;
}

relationship type "System-FocusSystem passing"
{
    line type solid;
    directed;
    line width 0;
}

relationship type "Composition"
{
    line type solid;
    line width 2;
}

/* ROLE DEFINITIONS */

role type "Client system" of "FocusSystem-System passing"
{
    symbol "Line";
    connectivity (0,M);
    properties ("Constraints");
}
role type "Server system" of "FocusSystem-System passing"
{
  symbol "Arrow";
  connectivity (0,M);
  properties ("Signature","Sequence number", "Status");
}
ole type "Client system" of "System-FocusSystem passing"
{
  symbol "Line";
  connectivity (0,M);
  properties ("Constraints");
}

role type "Server system" of "System-FocusSystem passing"
{
  symbol "Arrow";
  connectivity (0,M);
  properties ("Signature","Sequence number", "Status");
}

role type "Client system" of "Composition"
{
  symbol "Solid Line";
  connectivity (0,M);
}

role type "Server process" of "Composition"
{
  symbol "Solid Line";
  connectivity (0,M);
}

/* BINDING DEFINITIONS */

bind relationship "FocusSystem-System passing"
as "Client system" ("Focus System"),
as "Server system" ("System")
bind relationship "System-FocusSystem passing"
as "Client system" ("System"),
as "Server system" ("Focus System")
bind relationship "Composition"
as "Client system" ("Focus System"),
as "Server process" ("Process")
Method Definition File of the SED Metamodel

(Refer to the equivalent OPPR specification in Section 5.15, Figure 5.9).

/* Author   : Vasuthevan Balakrishnan */
/* Project  : HYCONAN - Graphics Editor */
/* Date Completed : 15/4/1999 */

metamodel "HYCONAN State Evolution Diagram (SED)"
extension "SED"

/* SHAPE DEFINITIONS */

shape "Rectangle"
{ shape
  (0@10,5@15,10@0,190@0,195@5,200@10,200@190,195@195,190@200,
   10@200,5@195,0@190);
  connection points (0@0,200@0,200@200,0@200);
  line type solid;
  line width 1;
  color black filled white;
}

shape "Arrow"
{ shape (100@100,90@95,90@105);
  line type solid;
  line width 0;
  color black filled black;
}

shape "Invisible Rectangle"
{ shape (0@0,200@0,200@50,0@50);
  line type invisible;
  connection points (0@0,200@0,200@200,0@200);
}

shape "Line"
{ shape (95@100,100@100);
  line type solid;
  line width 1;
  color black;
}

/* SYMBOL DEFINITIONS */

symbol "Rectangle"
{ shapes ("Rectangle);
  scale 0.5;
labels {"Name" at (1 1 200 25) centered;
   "Kind" at (1 26 200 50) centered;
   "Event if Start" at (1 51 200 75) centered;
   "Activity" at (1 76 200 100) centered;
}

symbol "Invisible Rectangle"
{ shapes ("Invisible Rectangle");
  scale 0.5;
  labels {"Related item" at (1 1 200 25) centered;
   "Detail" at (1 26 200 50) centered;
  }
}

symbol "Line"
{ shapes ("Line");
  scale 1.0;
  labels {"Event" at (45 105 90 130) centered;
   "Conditions" at (45 70 90 95) centered;
  }
}

symbol "Arrow"
{ shapes ("Arrow");
  scale 1.0;
  labels {"Action" at (45 70 90 95) centered;
   "Operation" at (45 105 90 130) centered;
  }
}

/* PROPERTY DEFINITIONS */

property type "Name"
{ datatype String;
  values unique;
}

property type "Kind"
{ datatype editlist("{Start}","{Stop}","{Iterative}");
}

property type "Event if Start"
{ datatype String;
}
property type "Activity"
{ datatype editlist ("Ac: ");
}

property type "Recursive transitions"
{ datatype Text;
}

property type "Event"
{ datatype String;
}

property type "Conditions"
{ datatype editlist ("{ } ");
}

property type "Action"
{ datatype String;
}

property type "Operation"
{ datatype editlist (" ");
}

property type "Related item"
{ datatype String;
  values unique;
}

property type "Detail"
{ datatype Text;
}

/*@ OBJECT DEFINITIONS */

object type "Note"
{ symbol "Invisible Rectangle";
  duplicates allowed;
  properties ("Related item", "Detail");
}

object type "State"
{ symbol "Rectangle";
duplicates allowed;
properties ("Name", "Kind", "Event if Start", "Activity", "Recursive transitions");
}

/* RELATIONSHIP DEFINITIONS */

relationship type "Transition"
{
    line type solid;
directed;
    line width 0;
}

/* ROLE DEFINITIONS */

role type "Source state" of "Transition"
{
symbol "Line";
    connectivity (0,M);
    properties ("Event", "Conditions");
}

role type "Destination state" of "Transition"
{
    symbol "Arrow";
    connectivity (0,M);
    properties ("Action","Operation");
}

/* BINDING DEFINITIONS */

bind relationship "Transition"
as "Source state" ("State"),
as "Destination state" ("State")