Notes On Object-Oriented Modeling And Design

Article · January 1995

Source: CiteSeer

CITATIONS
5

READS
29,390

1 author:

Stephen Clyde
Utah State University
48 PUBLICATIONS **135** CITATIONS

SEE PROFILE

NOTES ON OBJECT-ORIENTED MODELING AND DESIGN

Stephen W. Clyde

Brigham Young University
Provo, UT 86402

Abstract: A review of the Object Modeling Technique (OMT) is presented. OMT is an object-oriented method described by Rumbaugh, et. al. in the book *Object-oriented Modeling and Design*. These notes provide a summary of OMT, as well as a list of its strengths and weaknesses. In addition, a thorough description of OMT models and the OMT methodology is provided in the Appendix. This description is in the form of an *Object-oriented Systems Analysis* (OSA) model.

1. INTRODUCTION

The Object Modeling Technique (OMT) is an object-oriented analysis, design, and implementation methodology that focuses creating a model of objects from the real world and then using this model to develop object-oriented software. OMT was developed by James Rumbaugh, et. al., at General Electric's Research and Development Center. The comments in this review are based on their book, *Object-oriented Modeling and Design* [Rumbaugh-1991].

In general, the software engineering community has not yet come to a consensus on the meaning of "object-oriented". So like most authors, Rumbaugh and his co-authors introduce their ideas with a discussion on what they believe are the important characteristics of an object-oriented approach. The following quotes summarize their perspective:

> Superficially the term "object-oriented" means that we organize software as a collection of discrete object that incorporate both data structure and behavior.... There is some dispute about exactly what characteristics are required by an object-oriented approach, but they generally include four aspects: identity, classification, polymorphism, and inheritance. (pp. 1)

> The essence of object-oriented development is the identification and organization of application-domain concepts, rather than their final representation in a programming language. (pp. 4)

The rest of the book is organized into three parts. The first describes the various types of models used by the OMT methodology. The second part presents the OMT methodology, which includes methods for analysis, system design and object design. The third part discusses various implementation issue, such as establishing an appropriate programming style, working with

object-oriented languages, and working with non-object-oriented languages and using relational databases.

The following section provide notes of OMT modeling concepts and the OMT methodology. Sections 3 and 4 lists the strengths and weaknesses of OMT. The Appendix contains a complete OSA model of OMT modeling concepts and the OMT methodology.

2. OMT SUMMARY

2.1 OMT Models

The OMT methodology uses three kinds of models to describe a system:

1. *Object model* - describes the objects in the system and their relationships;
2. *Dynamic model* - describes the interactions among objects in the system; and
3. *Functional model* - describes the data transformations of the system.

The components of each of these models is briefly described below.

2.1.1 Object model components

Object:

An object is a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand. All objects have identity and are distinguishable. pp 21

It is not clear whether or not an object can belong to more than one class. The solid triangle notation for generalization/specialization relationships means that the specializations are intersecting sets. This implies that objects can belong to more than one class. However, much of the discussion on objects, classes, and dynamic models, as well as the notation for object instances, leads the reader to believe that an object can belong to only class.

Class:

A class describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics (pp. 22).

Attributes and operations are specified for classes.

Class membership can be defined in two ways: implicitly by rule or explicitly by enumeration.

Classes can also be considered objects, and therefore, they can be classified and

generalized. A class can have class attributes and class operations (pp. 71)

Attribute:

An attribute is (represents) a data value which is held by the objects in a class. An attribute should be a pure data value, not an object. Pure data values do not have identity (pp. 23).

Operation:

An operation is a function or transformation that may be applied to or by objects in a class. The same operation may be defined for several different classes. However, the signature (i.e. output type and formal parameter list) must be the same (pp. 25).

Method:

A method is the implementation of an operation for a class. The choice of which method is invoked it solely determined by the target object. The parameters are not a factor in determining which method is selected (pp. 25).

Link:

A link is a physical or conceptual connection between object instances. Links are order tuples (pp. 27).

Association:

An association describes a group of links with common structure and common semantics.

Attributes and operations can be specified for associations. Basically, this allows one to treat an association as a class (pp. 31).

Associations can have role names associated each class connection.

The links in an association can be ordered (pp. 35).

Associations can also have qualifiers on the class connections. qualifiers are special attributes that reduce the effective multiplicity (pp. 35).

Multiplicity:

Multiplicity specifies how many instances of one class may relate to a single instance of an associated class (pp. 30).

Multiplicity constraints can only be used with binary associations. Candidate keys have to be used in n-ary relationships.

Multiplicity constraints together with candidate keys are not as powerful as participation constraints and co-occurrence constraints.

Aggregation:

An aggregation is special type of association that represents a "part-of" relationship. Aggregations are transitivity and anti-symmetric (pp. 37).

Operations can be specified to propagate along aggregation hierarchies (pp. 60).

Generalization:

A generalization is a relationship between a class and one or more refined versions of that class. Generalization is transitive (pp. 39).

A generalization can have a *discriminator*, which is an attribute of an enumeration type that indicates which property of an object is being abstracted by the generalization (pp. 39)

A subclass may override a superclass feature by defining a feature with the same name (pp. 42). An operation may be overridden for one of the following reasons: extension, restriction, optimization, and convenience (pp. 64).

A generalization is not considered an association.

A *concrete class* is a class that will have members. An *abstract class* will not have any direct instances. An abstract class can not be a leaf class on a generalization hierarchy (pp. 61).

An instance of a class is an instance of all ancestors of the class (pp. 63).

A subclass can not change the signature of superclass's operation (pp. 63).

A subclass may add new operations and attributes. This is called *extension* (pp. 63).

A subclass may constrain an ancestor attribute. This is called *restriction* (pp. 63).

Inheritance:

Inheritance is a mechanism for sharing attributes and operations using a generalization relationship.

The dynamic model of a superclass is inherited by its subclasses. A subclass can have its own states. The state diagram for the subclass must be a refinement of the state diagram for the superclass (pp. 111).

Module:

A module is a logical construct for grouping classes, associations, and generalizations (pp. 43).

Sheet:

A sheet is portion of a model that fits on a single piece of paper (pp. 43).

Join Class:

A join class is a class that inherits features from more that one superclass. Conflict must be solved in the implementation (pp 65).

Metadata:

Metadata is data that describes other data. Metadata can be incorporated in a model to act a patterns for other data (pp. 69).

Instantiation:

An instantiation relates a class to an instance. A class may be an instance of a metadata class.

Constraints:

Constraints are functional relationships between entities of an object model. The term entity includes objects, classes, attributes, links, and associations (pp. 73).

Derived entities:

A derived object, link, or attribute is defined as a function of one or more objects, which in turn may be derived (pp. 75).

Homomorphisms:

A homomorphism is a mapping between two associations.

2.1.2 Dynamic model components

Events:

An event is something that happens at a point in time. It has no duration. Information can be passed on an event (pp. 85).

An event can cause a change in state. This is called an *event transition*. Event transitions can have conditions and actions associated with them.

Events can be classified and organized into a hierarchy (pp. 98).

The event hierarchy is independent of all generalization hierarchies (pp. 111).

States:

A state is an abstraction of the attribute values and links of an object (pp. 87).

A state corresponds to the interval between two events received by an object. A state has

duration (pp. 88).

A state is often associated with the value of an object satisfying some condition (pp. 88).

A state can have an activity, which is an action that take time (pp. 92).

A state can have entry, event and exit actions (pp. 101)

A state can have concurrent subparts (pp. 99).

State diagrams:

A state diagram relates states and events (pp. 89).

State diagrams can be nested. A single state can have subdiagrams associated with it (pp. 94).

A nested state diagram is actually a form of generalization on states (pp. 96).

2.1.3 Functional model components

Action:

An action is a type of operation (pp. 131).

An action is a transformation that has side effects on the target object or other objects in the system reachable from the target object (pp. 131).

An action has no duration in time; it is logically instantaneous (pp. 131).

Activity:

An activity is a type of operation (pp. 131).

An activity is an operation to or by an object that has duration in time (pp. 132).

Actor:

An actor is an active object that drives the data flow graph by producing or consuming values (pp. 126).

Actors are explicit objects in the object model. Data flows to or from actors represent operations on or by the objects (pp. 138).

Client:

A client object is the target object of an operation (pp. 138).

Constraint:

A constraint shows the relationship between two objects at the same time or between two values of the same object at different times (pp. 132).

A constraint between values of an object over time is called an *invariant* (pp. 133).

Control Flow:

A control flow is a Boolean value that affects whether a process is evaluated (pp. 129).

Data flow diagram:

A data flow diagram shows the functional relationships of the values computed by a system, including input values, and output values, and internal data stores. A data flow diagram contains *processes* that transform data, *data flows* that move data, *actor* objects that produce and consume data, and *data store* objects that store data passively. pp 124

Data flows can copy values to multiple destinations (pp. 126)

Aggregate data flows can be split into the sub-parts (pp. 126).

Data flow diagrams can be nested. This is also called *leveling*.

Data store:

Data stores are passive objects that respond to queries and updates. Data stores are also objects or attributes in the object model (pp. 138).

Data stores do not generate operations on their own but merely responds to requests to store and access data (pp. 127).

Operations:

Each leaf process is an operation (pp. 130). Each operation may be specified in various ways, including the following:

- mathematical functions;
- table of input and output values;
- equations specifying output in terms of input;
- pre- and post-conditions;
- decision tables;
- pseudocode;
- natural language.

The specification of an operation includes a signature and a transformation. The signature of all methods that implement an operation must match.

Non-trivial operations can be divided in three categories: queries, actions, and activities (pp. 131).

Processes:

A process transforms data values. The lowest-level processes are pure functions without side effects. A high-level process can be expanded into an entire data flow diagram.

Query:

A query is an operation that has no side effects on the externally visible state of any object (pp. 131).

A query with no parameters is called a *derived attribute* (pp. 131).

Signature:

A signature defines the interface to an operation; the arguments it requires and the values it returns.

2.1.4 The relationships of types of models to each other

Relative to the functional model:

- The object model shows the structure of the actors, data stores, and data flows.
- The dynamic model shows the sequence in which processes are performed (pp. 139).

Relative to the object model:

- The functional model shows the operations on the classes and the arguments of each operation. It therefore shows the supplier-client relationship among classes.
- The dynamical model shows the states of each object and the operations that are performed as it receives events and changes state (pp. 139).

Relative to the dynamic model:

- The functional model shows the definitions of the leaf actions and activities that are undefined with the dynamic model.
- The object model shows what changes state and undergoes operations (pp. 139).

2.2 OMT Methodology

The OMT methodology covers the full software development life cycle. However, the book only talks about analysis, design and some implementation. The methodology has the following stages:

1. *Analysis* - The analyst builds a model of the real-world situation showing its

important properties;

2. *System Design* - The system designer makes high-level decisions about the overall architecture, including the subsystem organization;
3. *Object Design* - The object designer builds a design model containing data structures and algorithms for object classes; and
4. *Implementation* - The object classes and relationships developed during object design are finally translated into a particular programming language, database, or hardware implementation (pp. 145).

2.2.1 OMT analysis

Analysis consists of iterating the following steps:

- generating a problem statement,
- building an Object Model,
- building an Dynamic Model,
- building an Functional Model,
- Finding operations.

The problem statement is description of the system requirements. It includes: problem scope, what is needed, application context, assumptions, and performance needs.

The step for constructing an object model as follows:

- Identify objects and classes.
  - Write down all object and classes that come to mind.
  - Remove redundant, irrelevant or bad object classes.
  - Some attribute identification is done at this level.
- Prepare a data dictionary.
- Identify associations between objects.
  - Write down all associations that come to mind.
  - Remove redundant, irrelevant or bad associations.
  - If possible, reduce n-ary associated by making associated object classes into link attributes.
  - Further specify association semantics.
- Identify attributes of objects and links.
  - Identify object and link attributes.
  - Remove unnecessary and incorrect attributes.
- Organize and simply object classes using inheritance.
  - Generalization/Specialization are used to specify inheritance, which is used as a re-use mechanism.
- Verify that access paths exist for likely queries.

- Iterate and refine the model.
- Group classes in modules.

Dynamic analysis begins with looking for event - externally-visible stimuli and responses (pp. 169).

For most problems, logical correctness depends on the sequences of interactions, not the exact times of interactions. Real-time systems are not addressed in this book (pp. 169).

The following steps are performed in constructing the dynamic model: (pp. 170)

- Prepare scenarios of typical interaction sequences.
  - major interactions
  - external display formats
  - information exchanges
  - special or exception cases
- Identify events between objects.
  - Events include all signals, inputs, decisions, interrupts, transitions, and actions to or from users to external devices (pp. 173).
  - An action by an object that transmits information is an event.
- Prepare an event trace for each scenario.
  - Event traces are ordered lists of events between objects (pp. 173).
  - Event flow diagrams show events between a group of classes without regard for sequence (pp. 173).
- Build a state diagram.
- Match events between objects to verify consistency.
  - Every event should have a sender and a receiver.
  - Paths starting with input events should match scenarios.

It is best to construct the functional model after the object model and dynamic model are constructed (pp. 180).

The following steps are performed in constructing a functional model: (pp. 180)

- Identify input and output values.
  - The input and output values are parameters of events between the system and the outside world.
- Build data flow diagrams showing functional dependencies.
- Describe functions.
- Identify constraints.
- Specify optimization criteria.

Adding operations is the next major step. Operations can correspond to queries about attributes or associations in the object, to events in the dynamic model, and to functions in the functional

model (pp. 183).

The key operations are summarized in the object model. They are discovered through the following step: (pp. 184)

- Find operations from the object model.
  - reading and writing attribute values
  - reading and writing association links
- Find operations from events.
  - Each event sent to an object corresponds to an operation.
  - The names of events are shown as the labels on the state transitions.
- Find operations from state diagram actions and activities.
- Find operations from functions.
  - Each function on the data flow diagram corresponds to an operation on an object(s).
- Look at "Shopping list" operations.
  - "Shopping List" operations are behaviors that are suggested by the real-world object, but not explicitly included in the problem statement.
- Simplifying Operations.
  - generalize
  - use inheritance

When the analysis is complete, the model serves as the basis for the requirements and defines the scope of future discourse (pp. 186).

The goal of the analysis is to fully specify the problem and application domain without introducing a bias to any particular implementation, but it is impossible in practice to avoid all taints of implementation (pp. 187).

This type of analysis involves some high level design activities.

2.2.2 OMT system design

During system design, decisions are made about how the problem will be solved. pp 199

System Design is the first design stage in which the basic approach to solving the problem is selected (pp. 199).

The *system architecture* is the overall organization of the system into components called *subsystems* (pp. 199).

During system design, the following decision have to be made:

- Organize the system into subsystems
- Identify concurrency
- 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

2.2.3 OMT object Design

During object design, the details of the models are fleshed out. Object design provides a basis for implementation (pp. 263). The following steps are preformed during object design: (pp. 263)

- Obtain operations for the object model from the other models.
  - Find an operation for each process in the functional model.
  - Define an operation for each event in the dynamic model.
- Design algorithms to implement operations.
- Optimize access paths to data.
- Implement software control by fleshing out the approach chosen during system design.
- Adjust class structure to increase inheritance.
- Design implementation of associations.
- Determine the exact representation of object attributes.
- Package classes and associations.

3. OMT STRENGTHS

- The models use fairly common ideas:
  - Object models are similar to ER models.
  - The dynamic model consists of state diagrams which are basic Harel charts.
  - The function model consists of data flow diagrams.

- The model constructs that offer good expressive power:
  - orderings on associations,
  - homomorphic mappings of associations,
  - derived Entities,
  - operation propagation through aggregations,
  - discriminators on generalizations,
  - patterns and instantiation relationships,
  - data dictionary,

- parameters on events, and
- entry and exit actions on states.

- The meta-model can be extended in a seamless manner.

4. WEAKNESSES

- The models and the method lack a theoretical foundation.

- The role and use of data flow diagrams seems to detract from the object-oriented approach.

- The syntax of the models are not well defined.
  - Sharing of model components among different models or versions of a model is not clear. For example, can a state diagram be part of more than one dynamic model?
  - It is not clear whether or not an object can belong to more than one class. The definition of an object instance seems to imply that an object can only belong to one class. However, since a non-overlapping properties can be specified for generalizations, it seems like objects can belong to more than one class.
  - Events are supposed to be instantaneous. However, the behavior of an event can be described by a state diagram. The semantics of such a diagram is not clear. Is the event an execution of the entire diagram or the turning on of a final state?
  - Module is not well defined. In some places it is described as set of classes. In other places it described as consisting of classes, associations and generalizations.
  - Are events labeled or are classes of events labeled?
  - The syntax of state subdiagrams is not well defined in relation to "do" activities.
  - The syntax of the global event flow diagrams is not well defined.
  - The syntax of the system design document is not well defined.

- The following model constructs are missing that would enhance expressive power:
  - High-Level Classes
  - High-Level Associations
  - Participation Constraints
  - Co-occurrence Constraints
  - Class Cardinality Constraints

- The meta-model contains redundancy.
  - Allowing attributes on an association is redundant with treating the

associations as class.

- To many design decisions are forced or biased during analysis
  - Attributes may be defined too early in the development process. (Note: it does seem that Rumbaugh encourages the delaying of attribute definition until after the association have been identified.)
  - The choice of data access paths and candidate keys may be too early.

- Encapsulation of operations and data seems weak, particularly when the data flow diagrams are use heavily. The specification of operations are spread across three different models.

- The splitting and synchronization of control in a state diagram is limited.

- Multiplicity constraints can not be generalized to n-ary association.

- The method tends to use state diagrams as little more than flow charts.

- Qualifiers on associations are not sufficiently expressive. Co-occurrence constraints on n-ary relationship sets are better. Also, the use of a qualifier may force the definition of an attribute to early.

- Generalization is not treated as a association. This adds to the complexity of the meta-model.

- The ending point for analysis, system design, and object design can be better defined in terms of what constitutes a completed Object Design Document.

REFERENCES

[1] David W. Embley, Barry D. Kurtz, Scott N. Woodfield, *Object-oriented Analysis: A Model-driven Approach*, Prentice Hall, 1992

[2] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-oriented Modeling and Design*, Prentice Hall, 1991

APPENDIX

An OSA model of OMT

The following figures contain an Object-oriented System Analysis (OSA) model for OMT. This OSA model is intended to assist someone familiar with OSA in understanding the details of OMT. It is not intended to be a substitute for reading *Object-oriented Modeling and Design*. Furthermore, this OSA model focuses on only the structure of OMT models and typically order of activities prescribed by the OMT method. It does not deal with notational conventions, model semantics, or method heuristics.

Figure Description	Figure No.
Analysis Document	
OMT Analysis Document Overview	1
Object Diagram: Objects and Links	2
Object Diagram: Classes and Associations	3
Object Diagram: Operations	4
Object Diagram: Attributes	5
Object Diagram: Aggregation	6
Object Diagram: Association Classes	7
Object Diagram: Generalization	8
Object Diagram: Modules and Sheets	9
Object Diagram: Abstract Classes and Concrete Classes	10
Object Diagram: Meta Data	11
Object Diagram: Candidate Keys	12
Object Diagram: Constraints	13
Class Diagrams and Instance Diagrams	14
Data Dictionary	15
State Diagram: States, Events, and Transitions	16
State Diagram: Internal Actions	17
State Generalization	18

Figure Description	Figure No.
Event Generalization	19
Concurrent Subdiagrams	20
Global Event Flow Diagrams	21
Data Flow Diagrams: Processes, Actors, Data Stores and Data Flows	22
Data Flow Diagrams: Control Flows	23
Data Flow Diagrams: Operations	24
Constraints	25
OMT Method: Analysis (Systems Analyst's Behavior)	
Analysis Overview	26
Develop Models	27
Develop Object Model	28
Develop Dynamic Model	29
Develop Functional Model	30
System Design Document	
System Design Document Overview	31
Basic System Architecture	32
OMT Method: System Design (Systems Designer's Behavior)	
System Design Overview	33
Object Design Document	
Object Design Document Overview	34
OMT Method: Object Design (Object Designer's Behavior)	
Object Design Overview	35

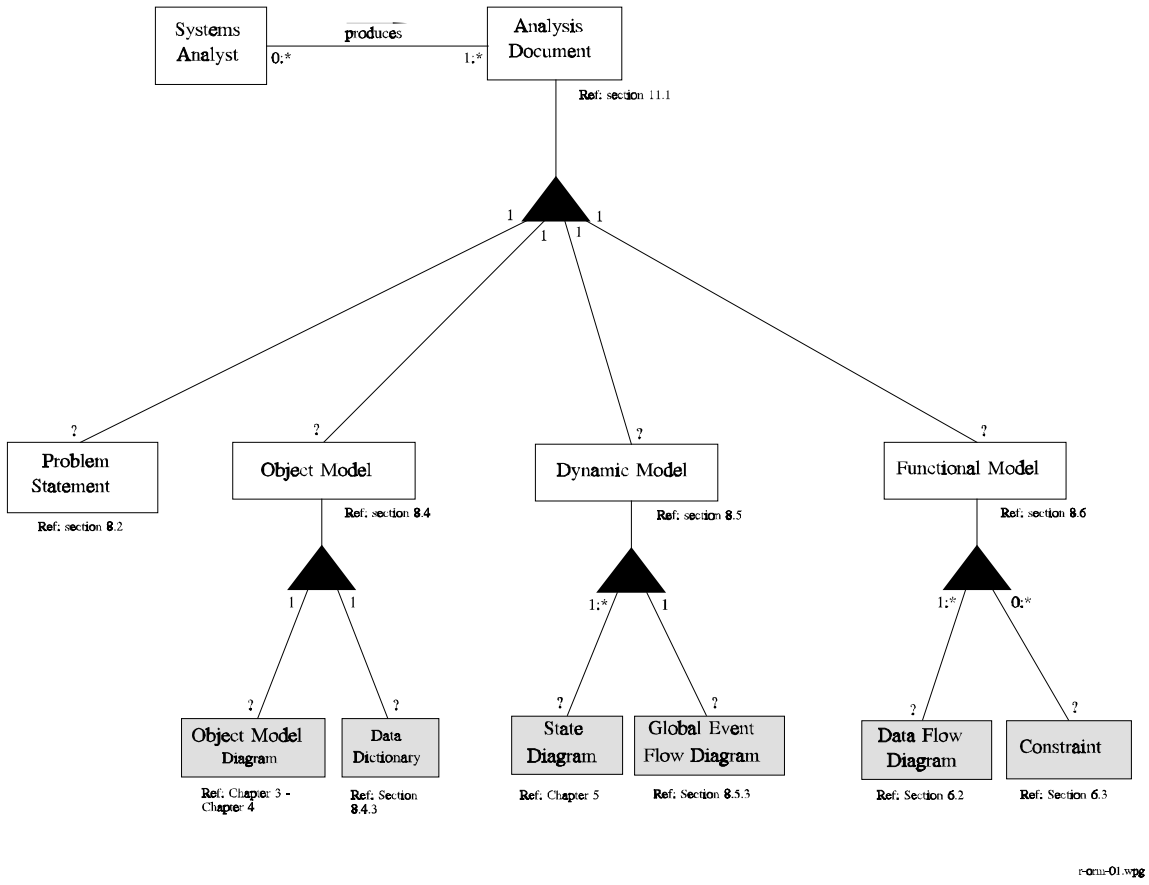


Figure 1

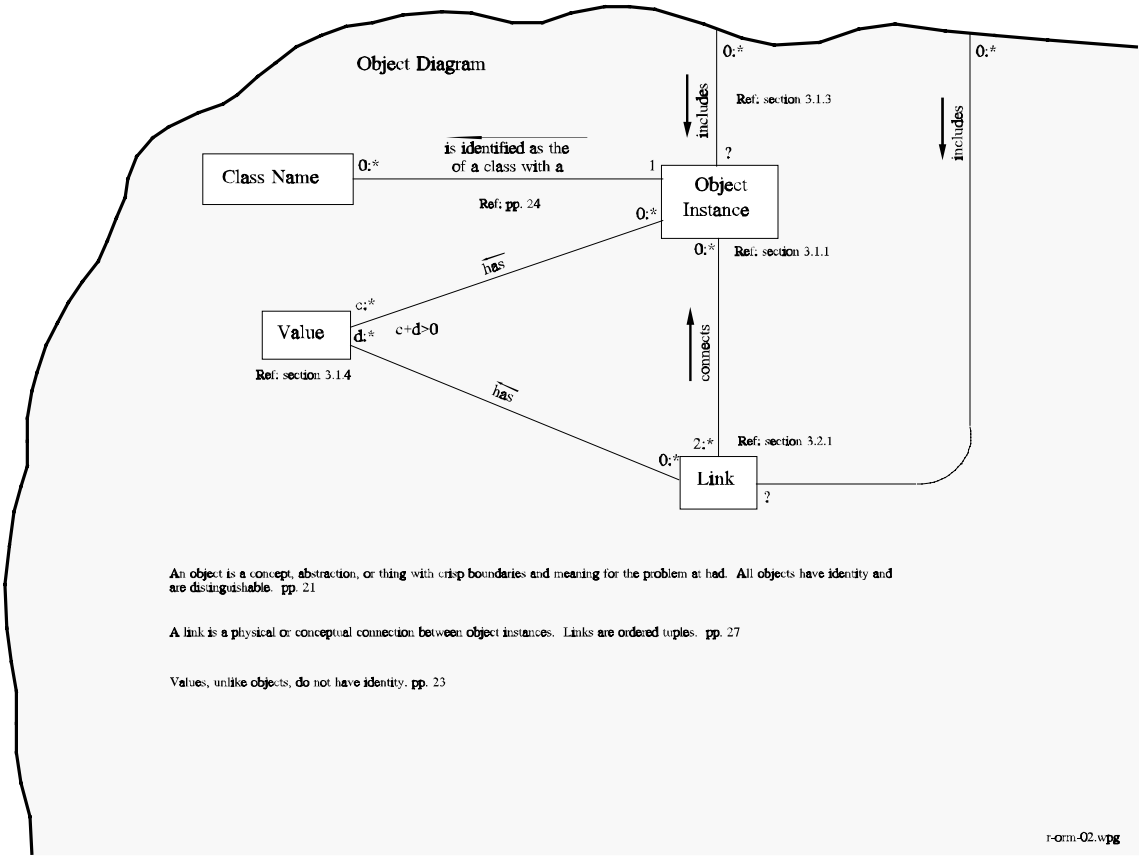


Figure 2

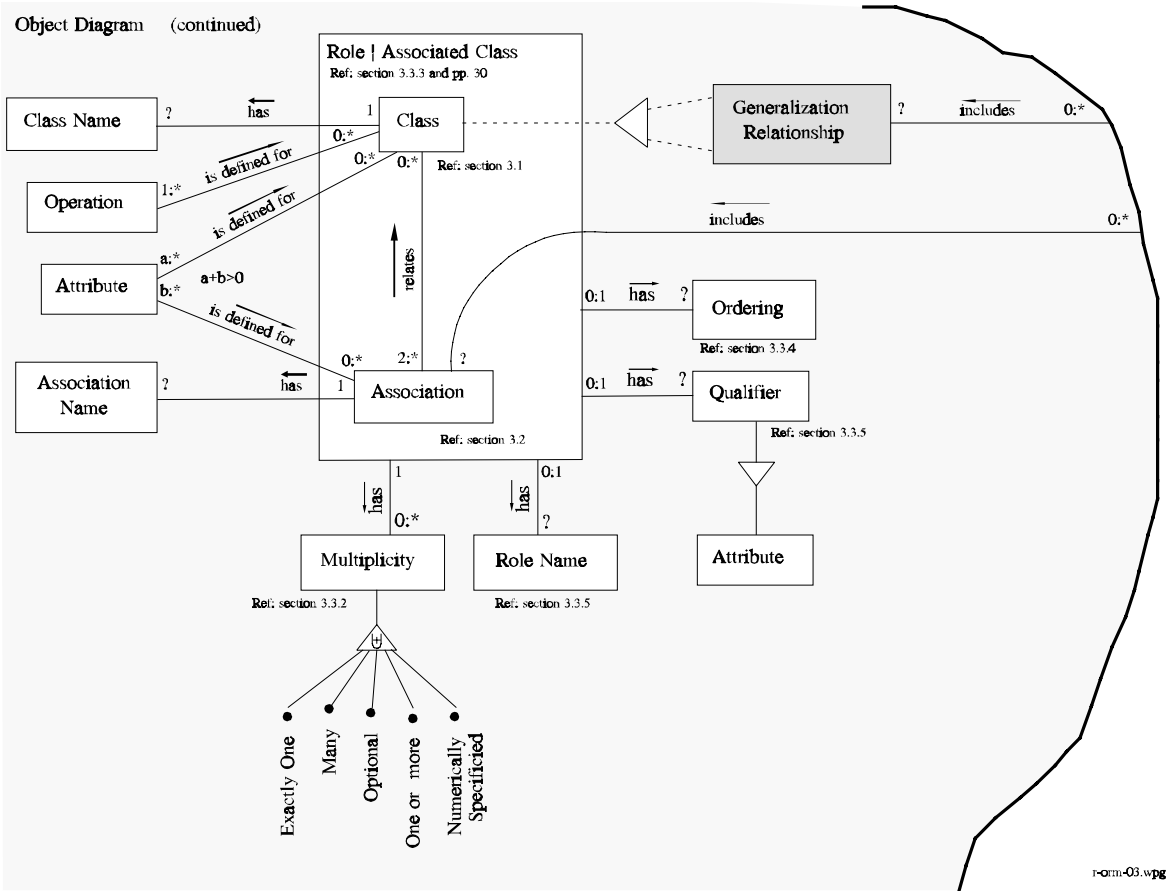


Figure 3

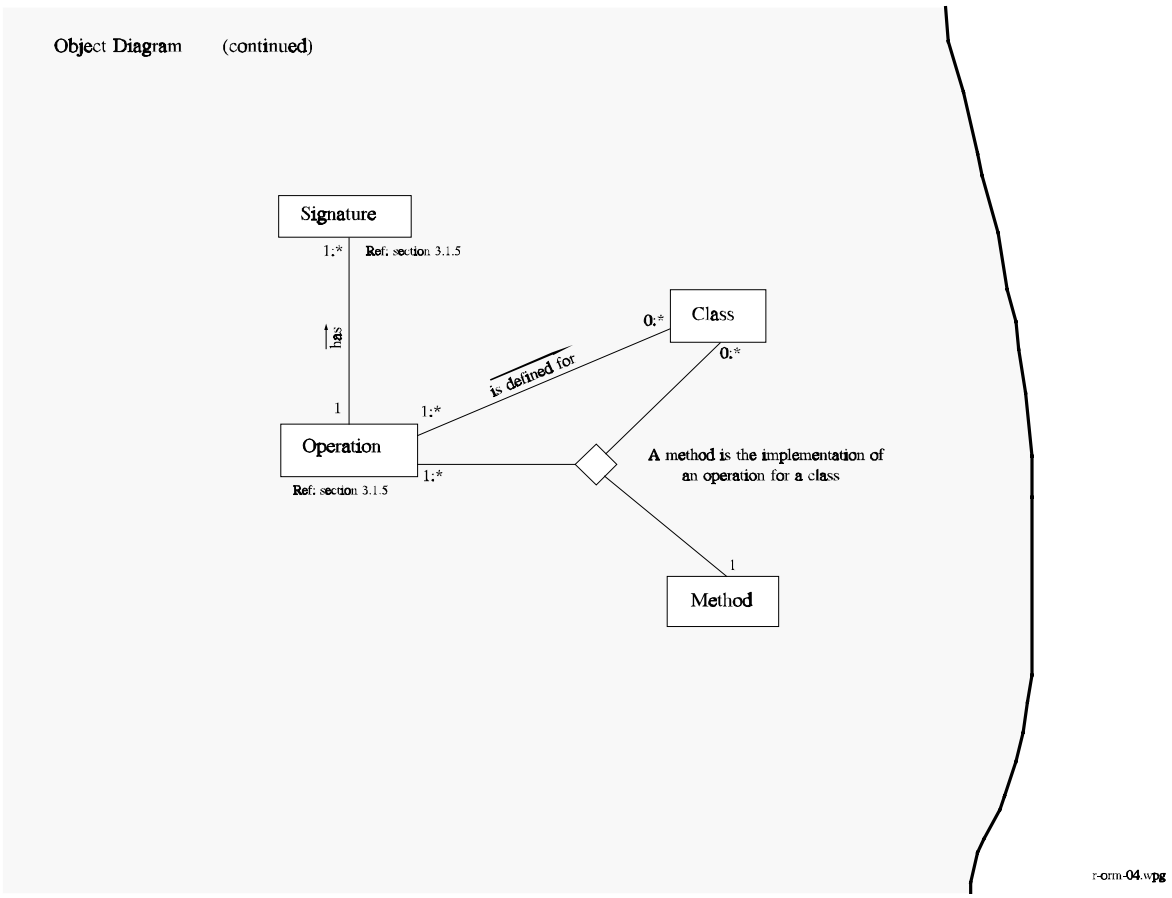


Figure 4

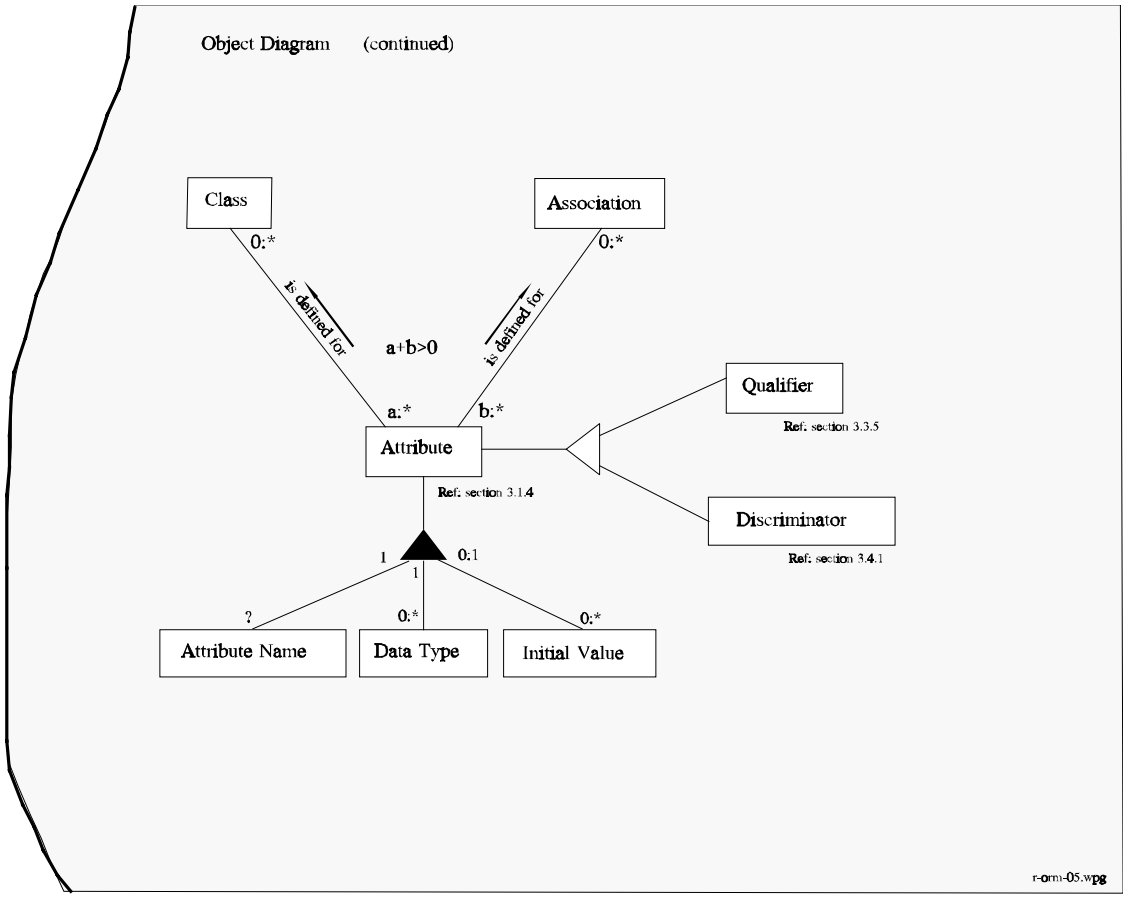


Figure 5

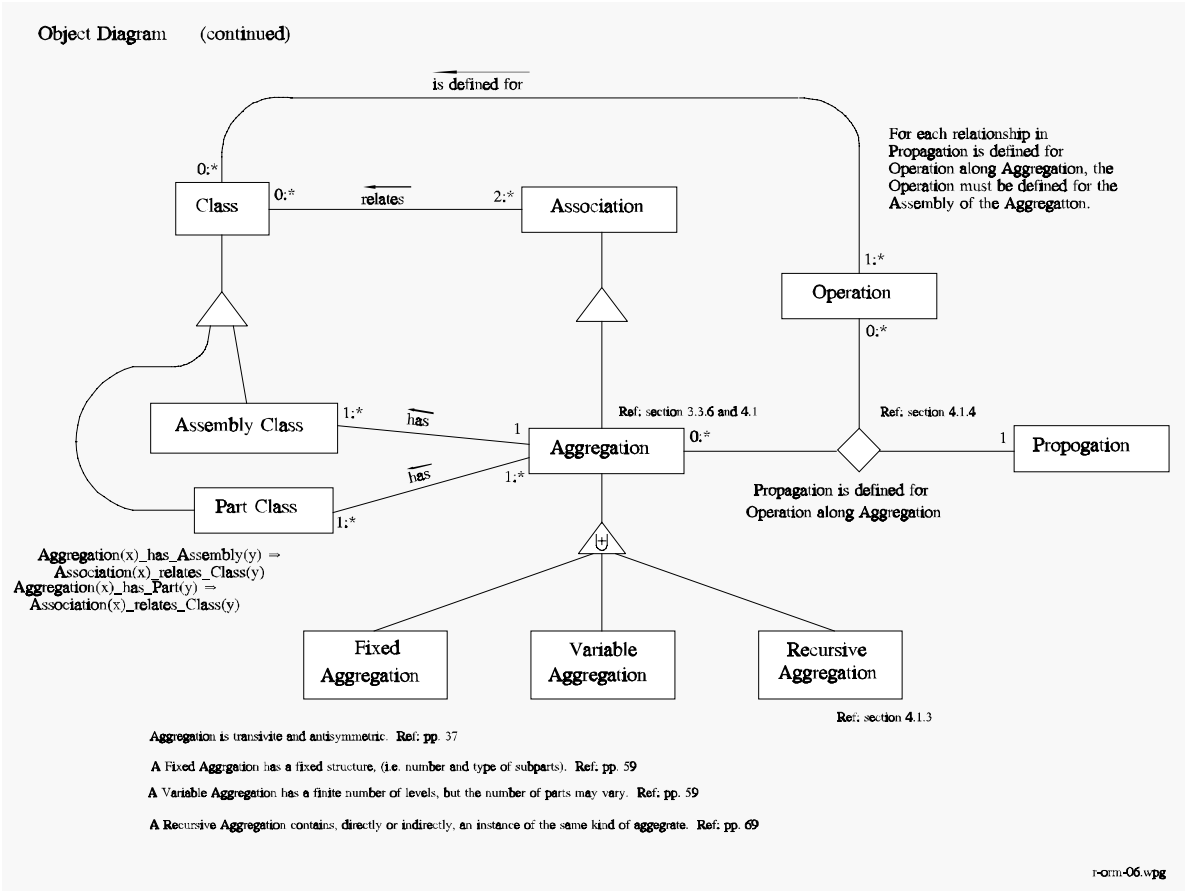


Figure 6

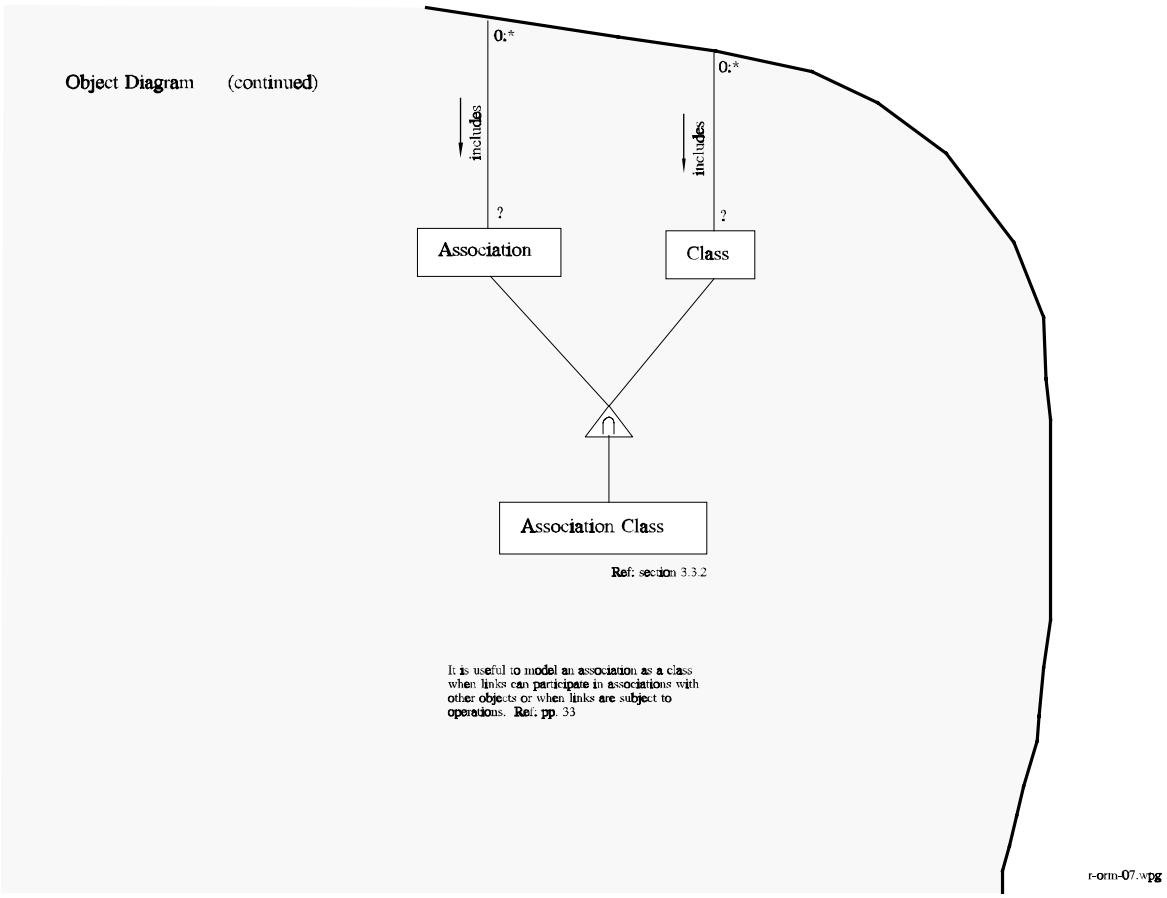


Figure 7

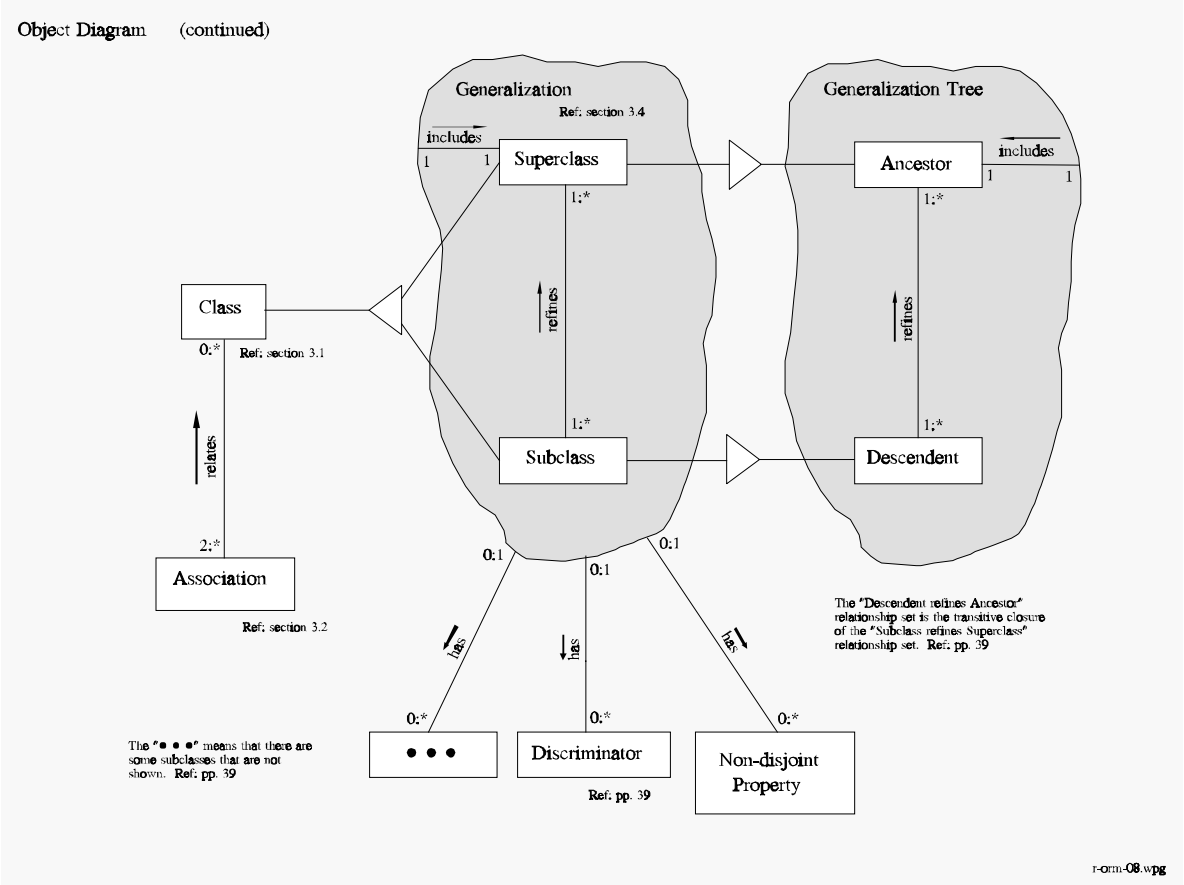


Figure 8

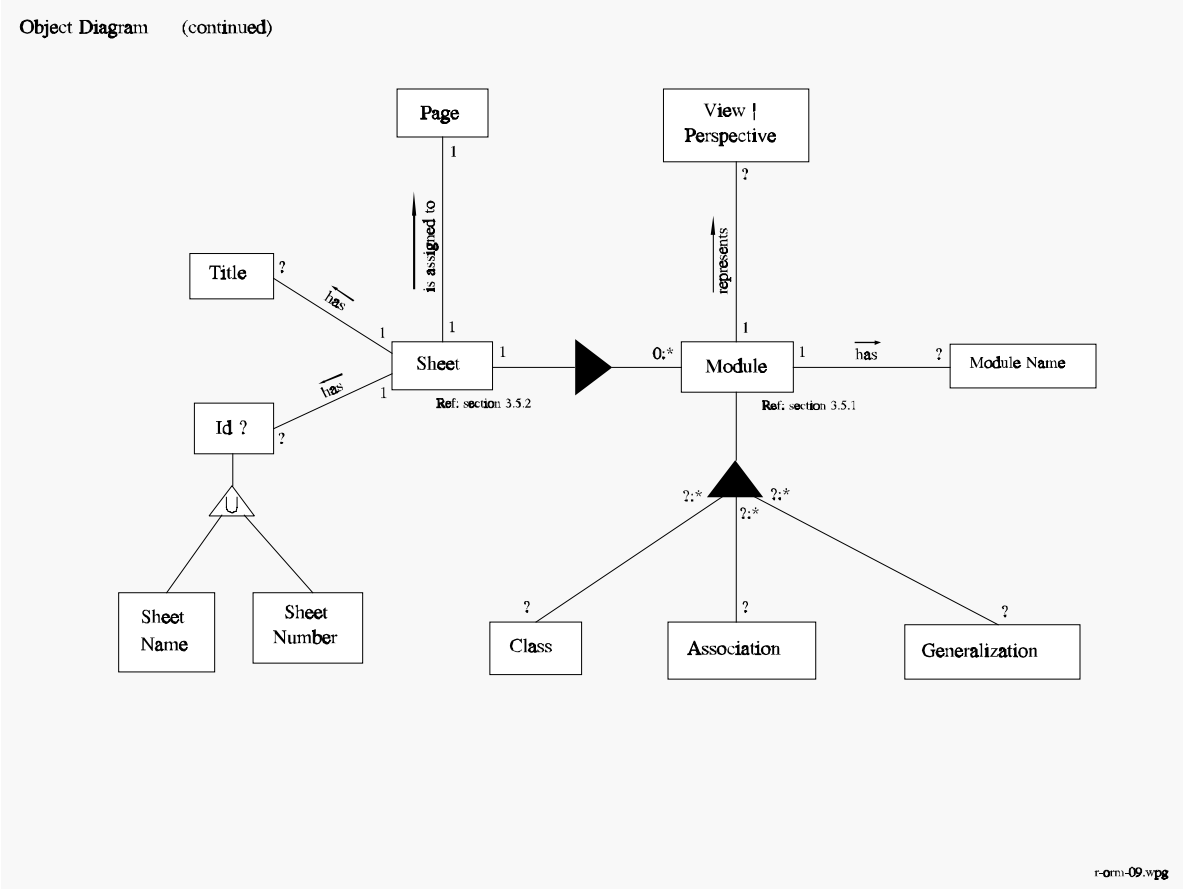


Figure 9

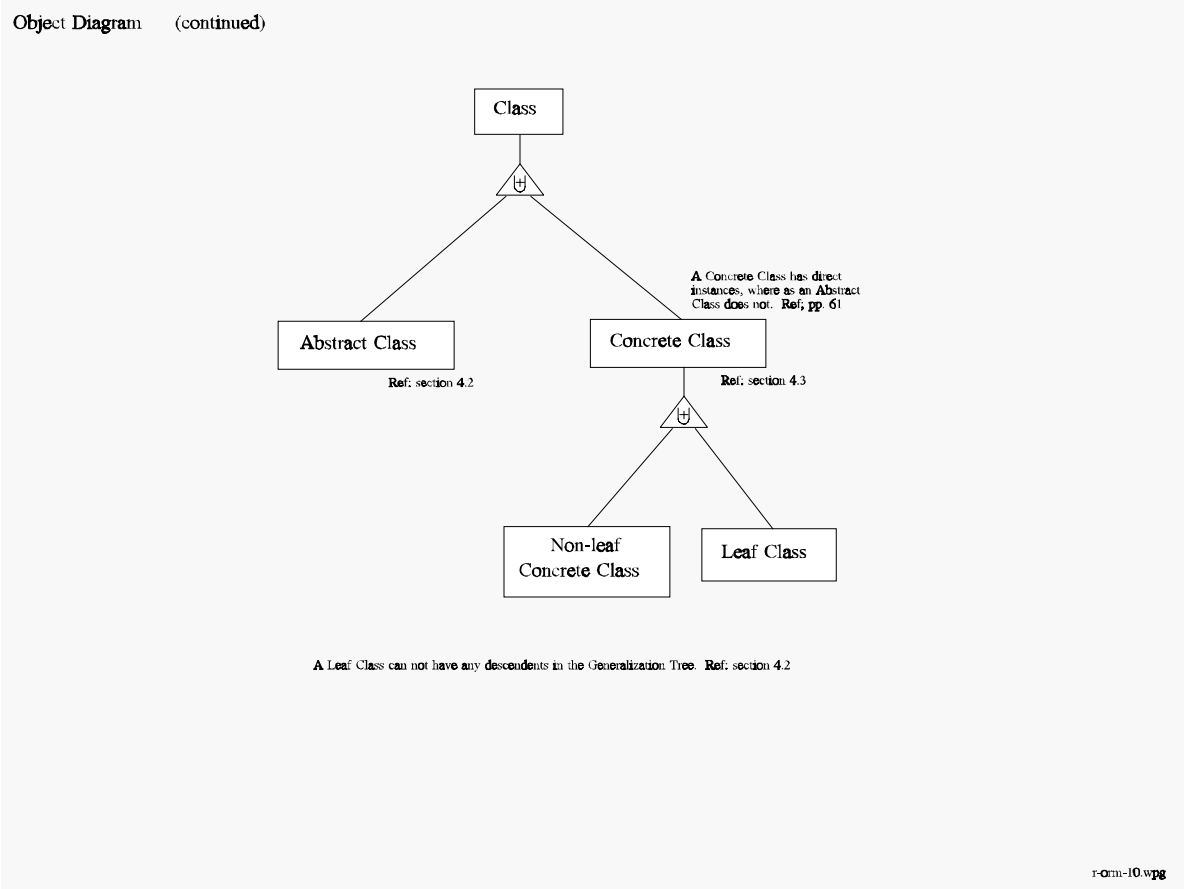


Figure 10

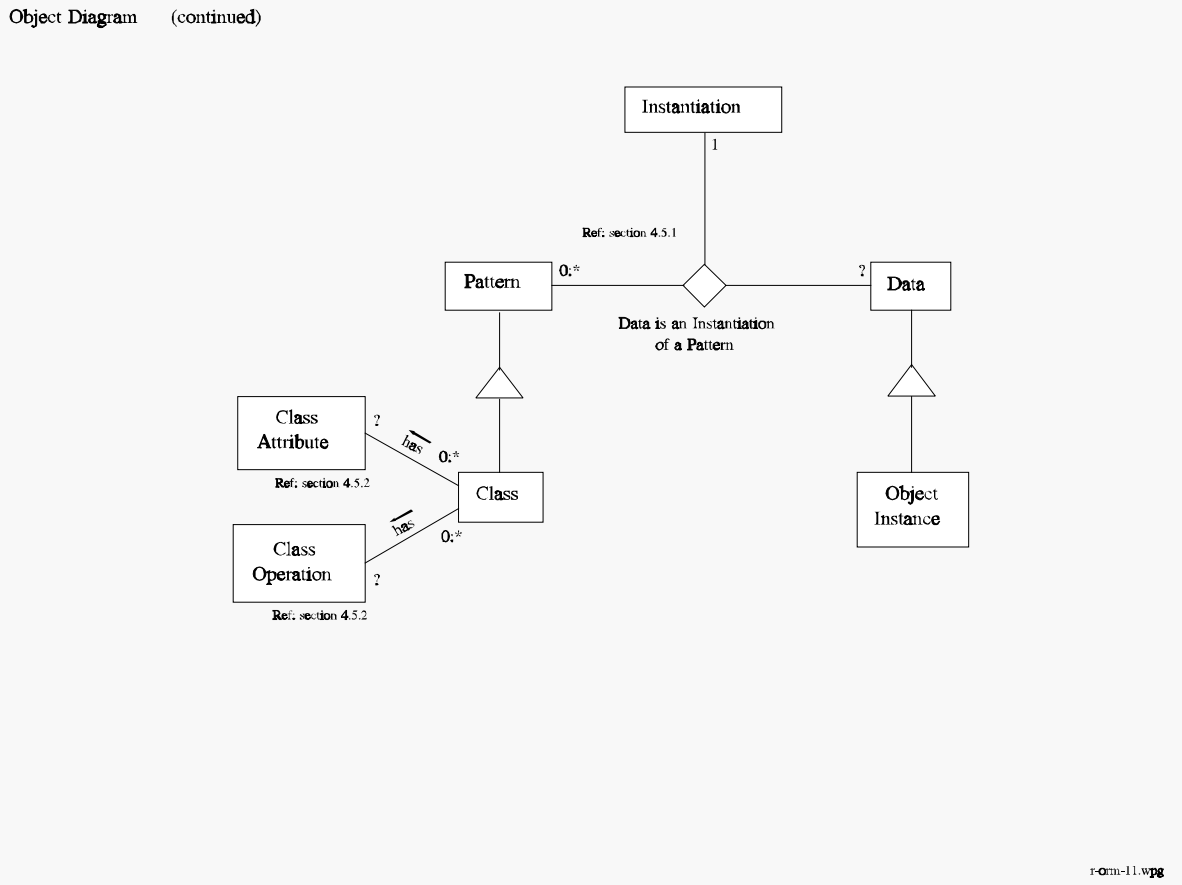
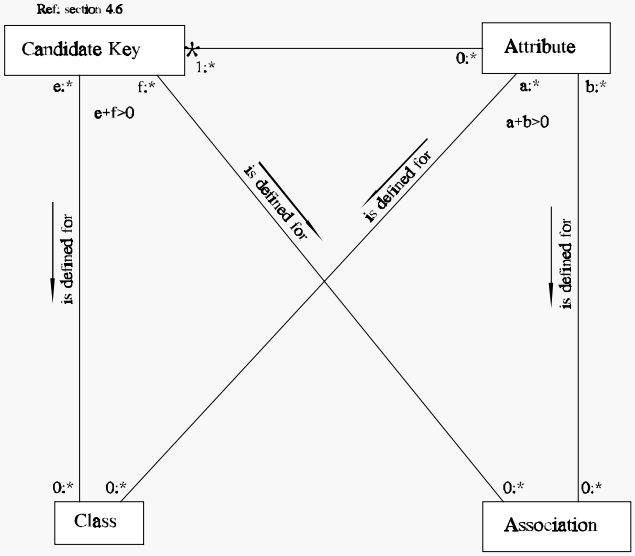


Figure 11

Object Diagram (continued)



A Candidate Key is a minimal set of attributes that uniquely identify objects and links in a Class or Association respectively. Ref: pp. 71

r-orm-12.wpg

Figure 12

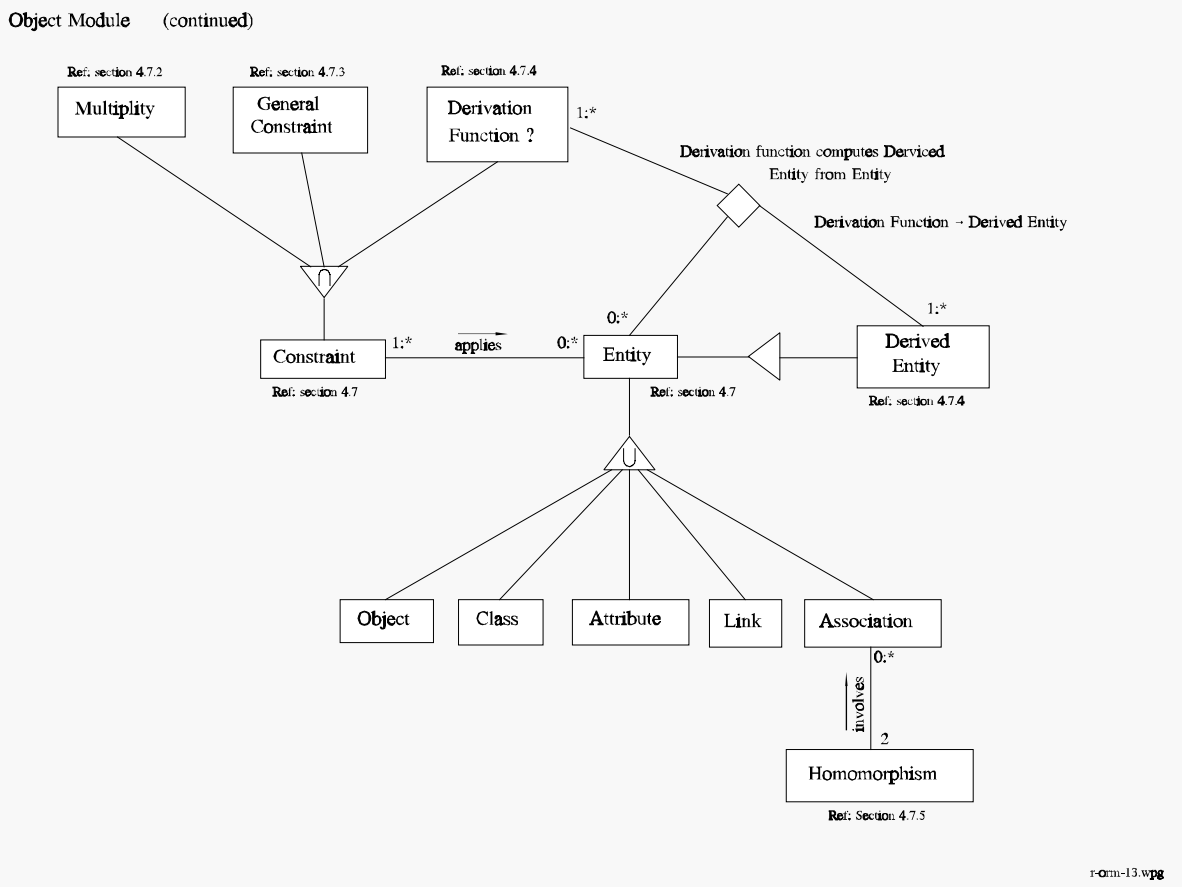
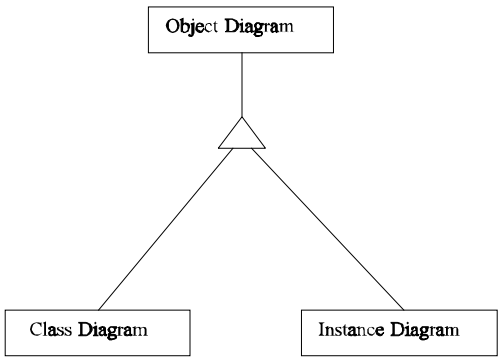
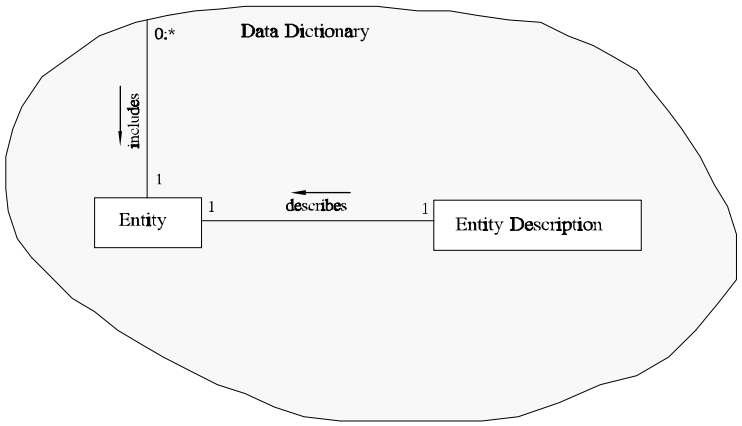


Figure 13



r-orm-14.wpg

Figure 14



Ref. section 8.5.3

r-orm-15.wpg

Figure 15

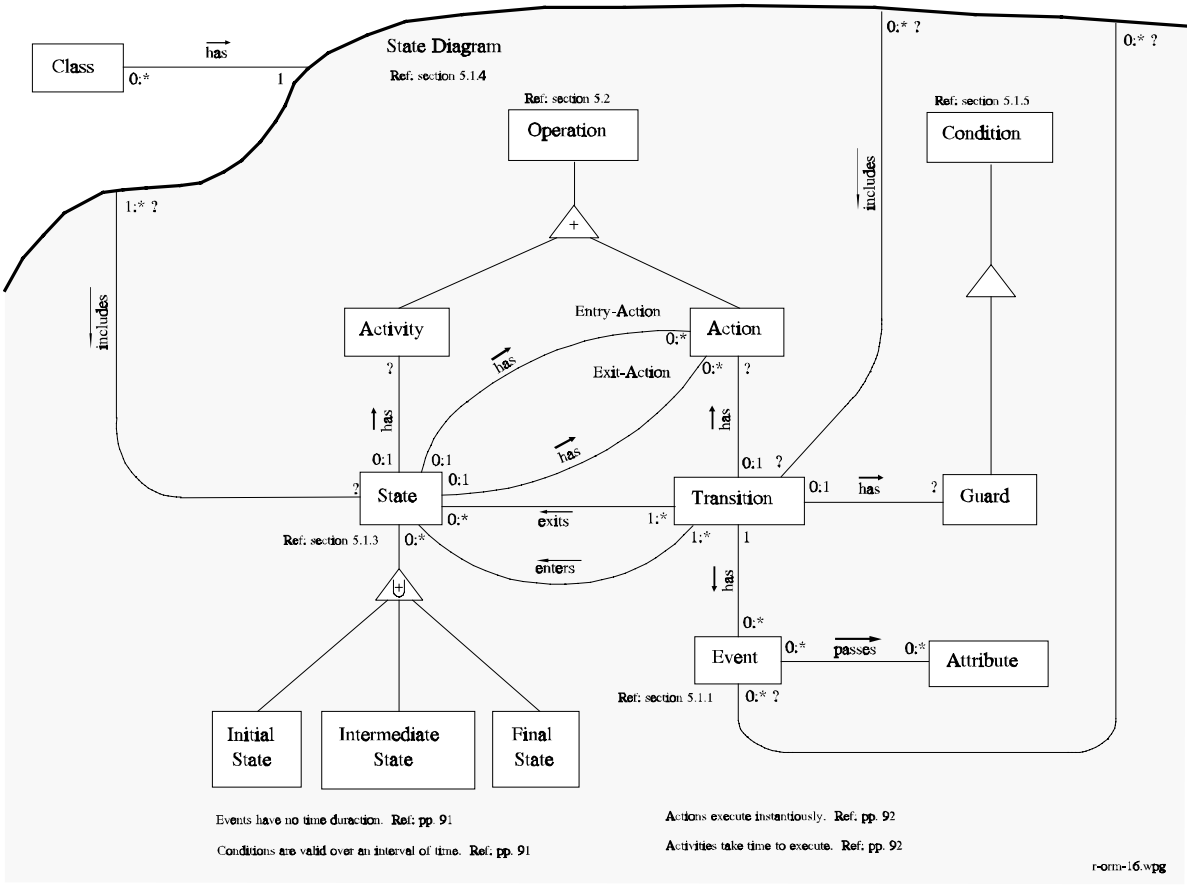


Figure 16

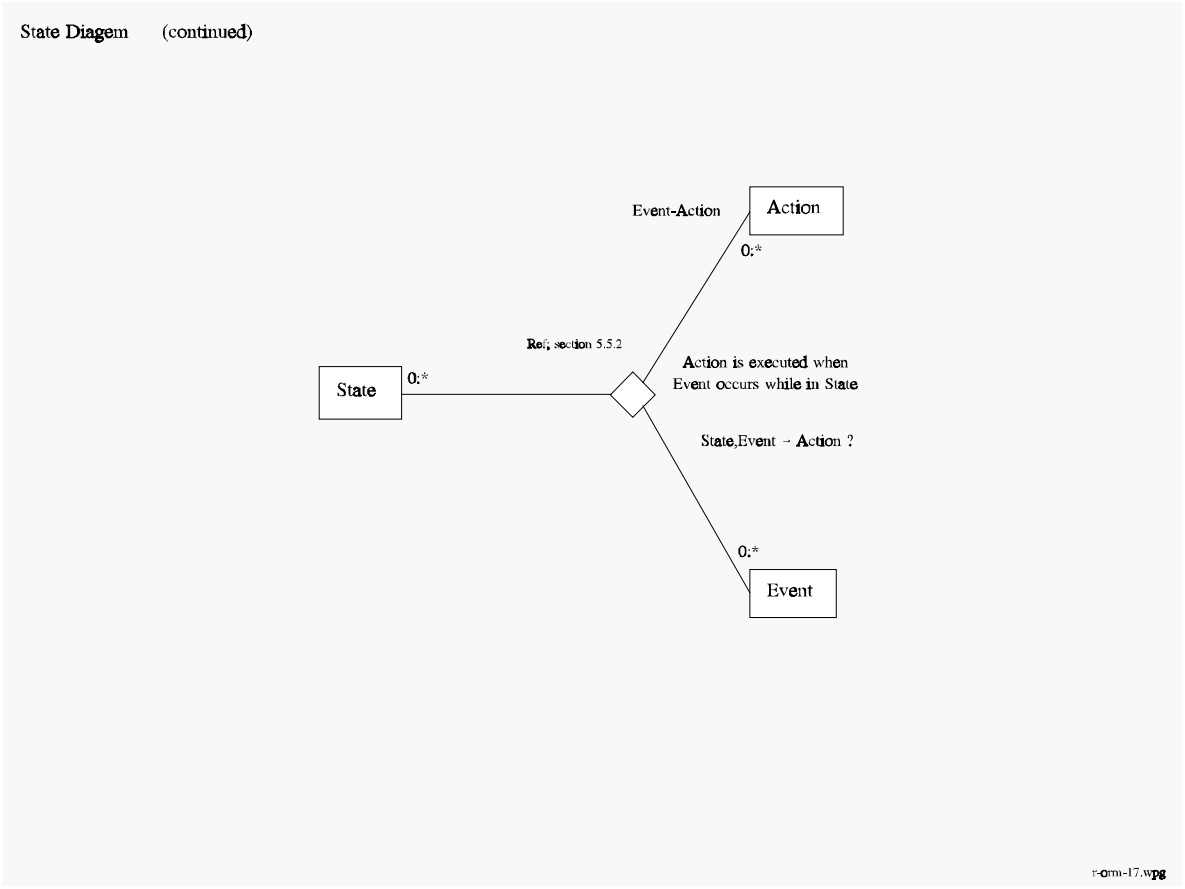
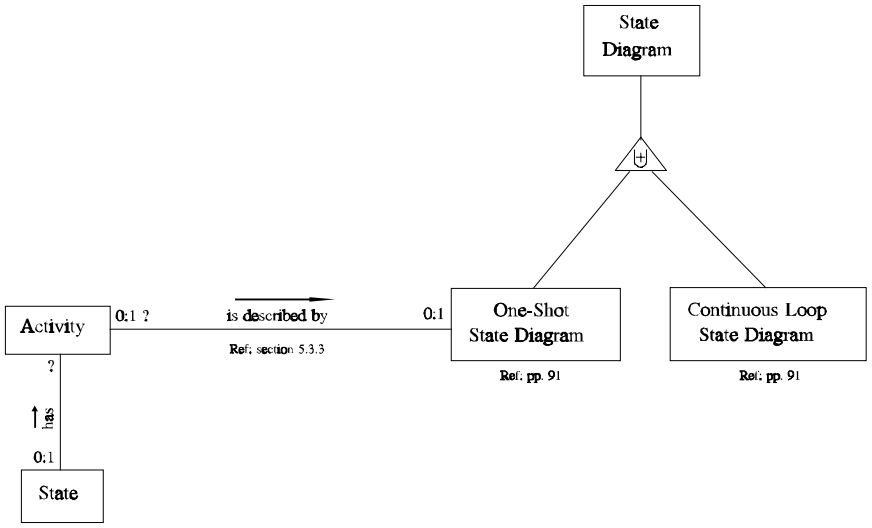


Figure 17



Generalization is equivalent to expanding nested activities. Ref: pp. 94

Activities, as well as events can be described with nested state diagrams. Ref: section 5.3

The set of nested diagrams forms a lattice. Ref: pp. 95

r-om-18.wpg

Figure 18

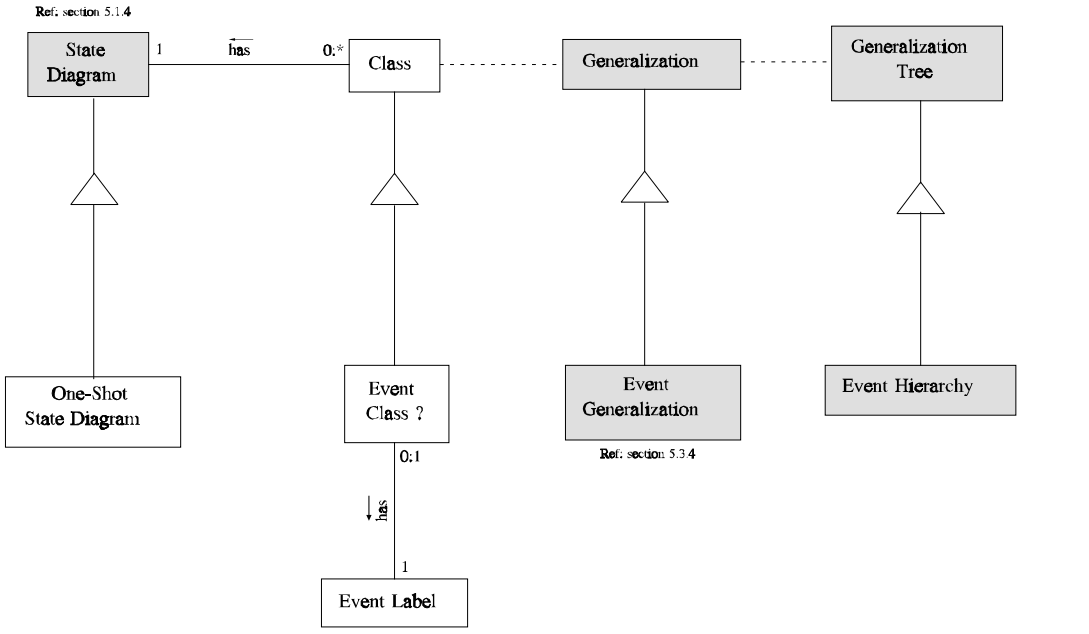
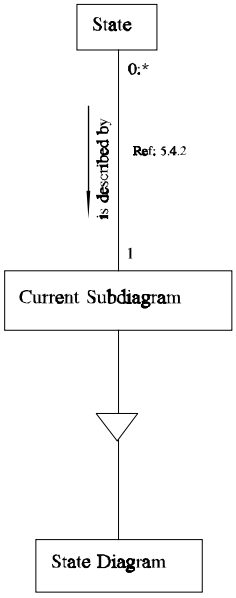
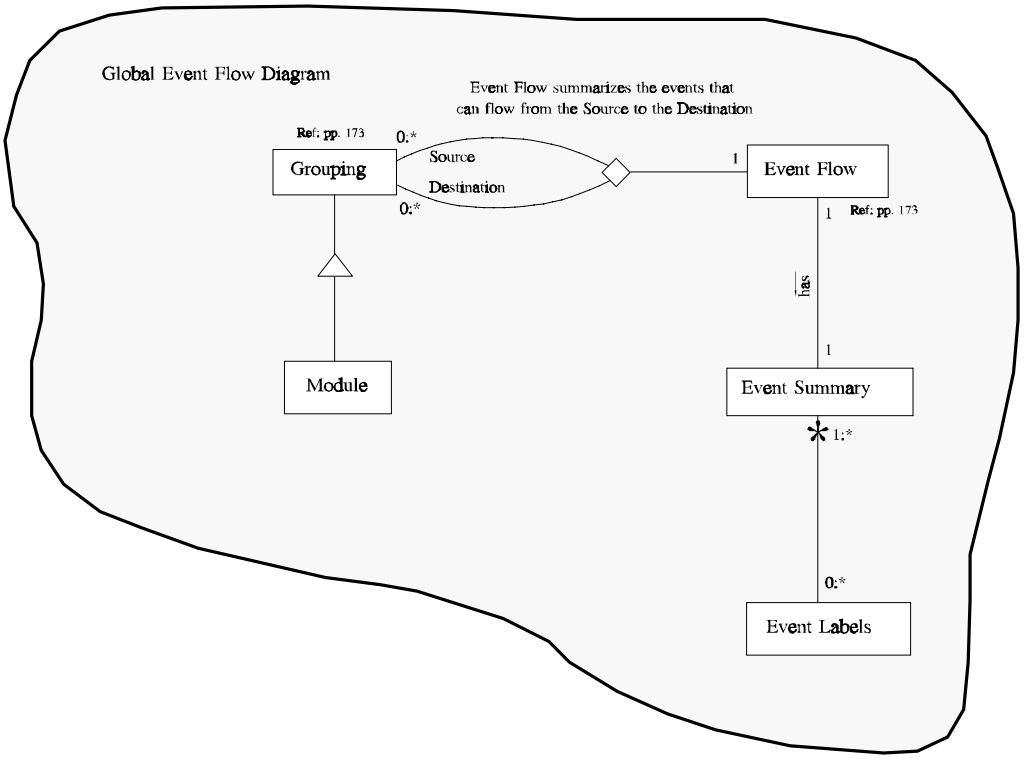


Figure 19



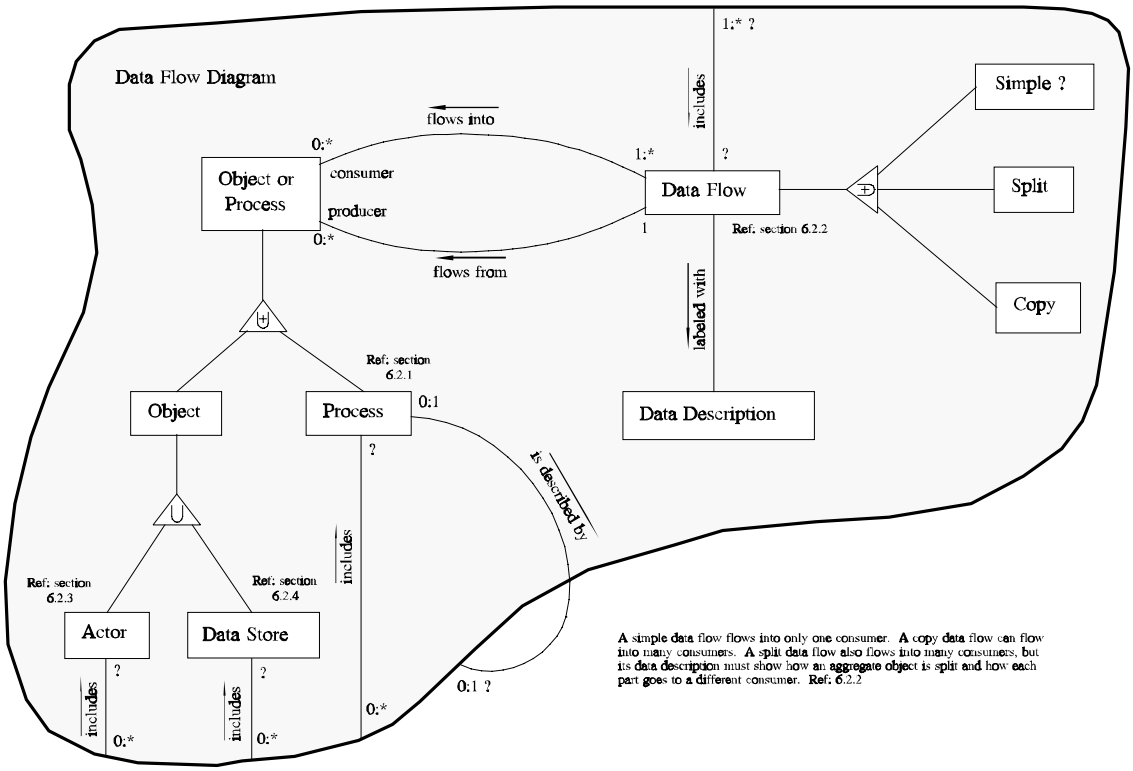
r-orm-20.wpg

Figure 20



r-orm-21.wpg

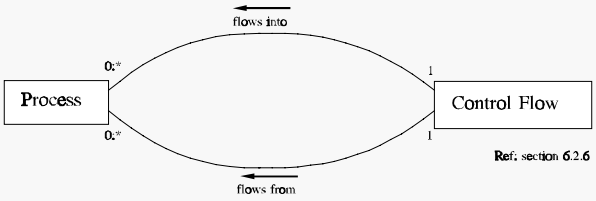
Figure 21



r-orm-22.wpg

Figure 22

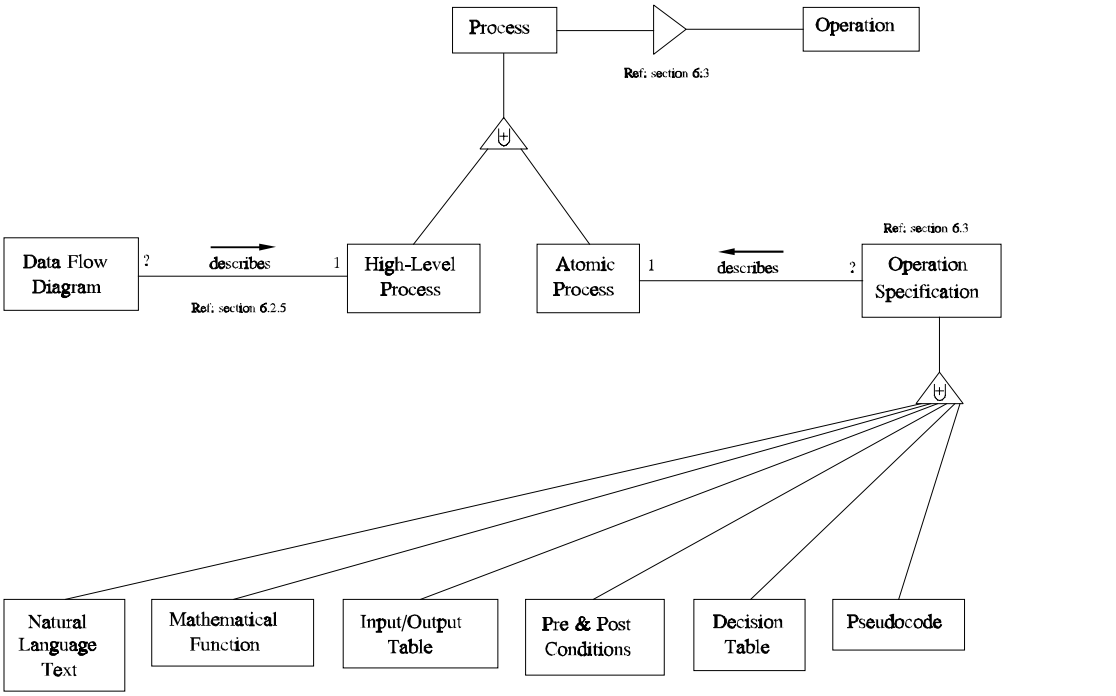
Data Flow Diagram (continued)



The use of Control Flows is discouraged. Ref. pp. 129

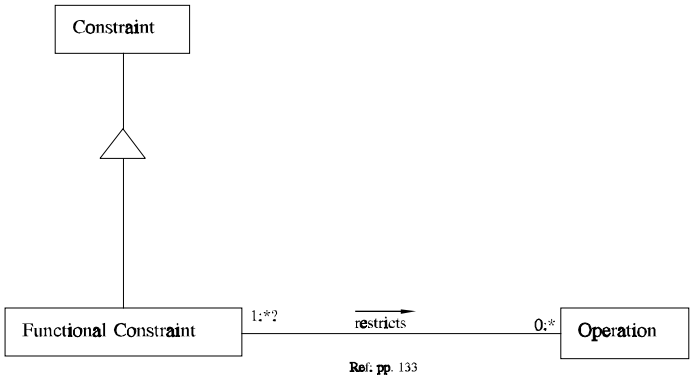
r-orm-23.wpg

Figure 23



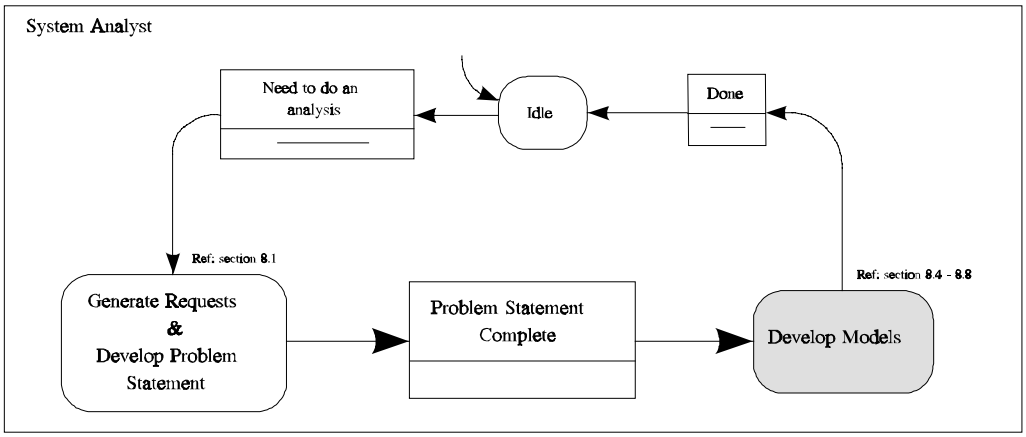
r-orm-24.wpg

Figure 24



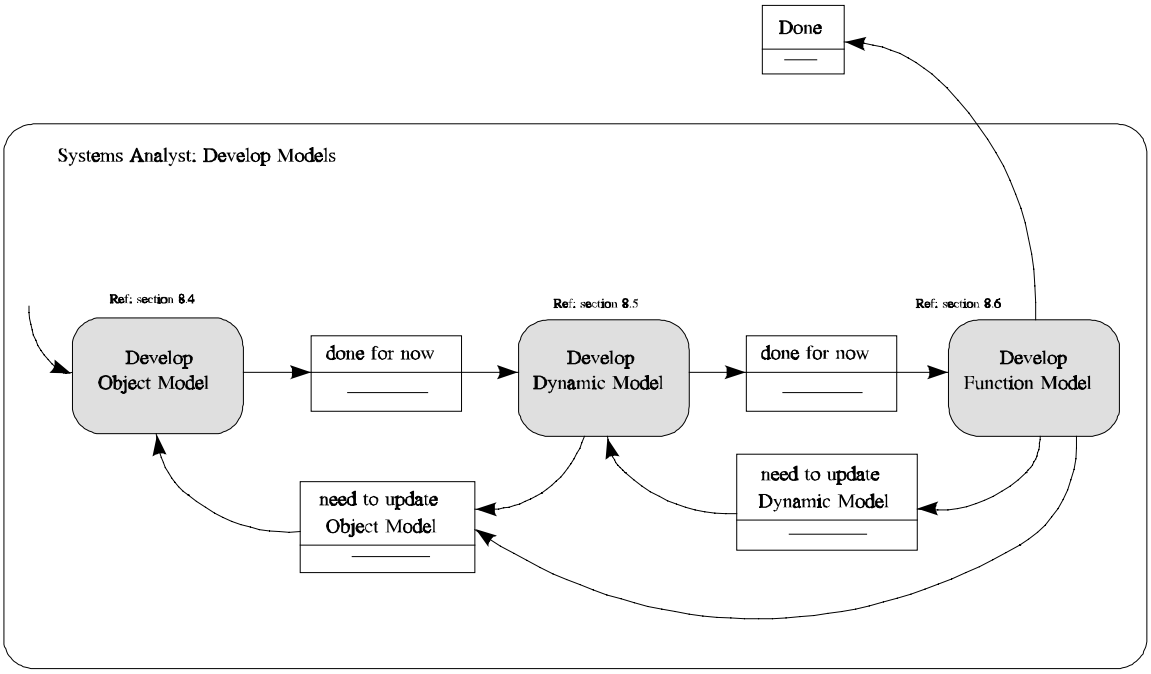
r-orm-25.wpg

Figure 25



r-obm-1.wpg

Figure 26



r-obm-02.wpg

Figure 27

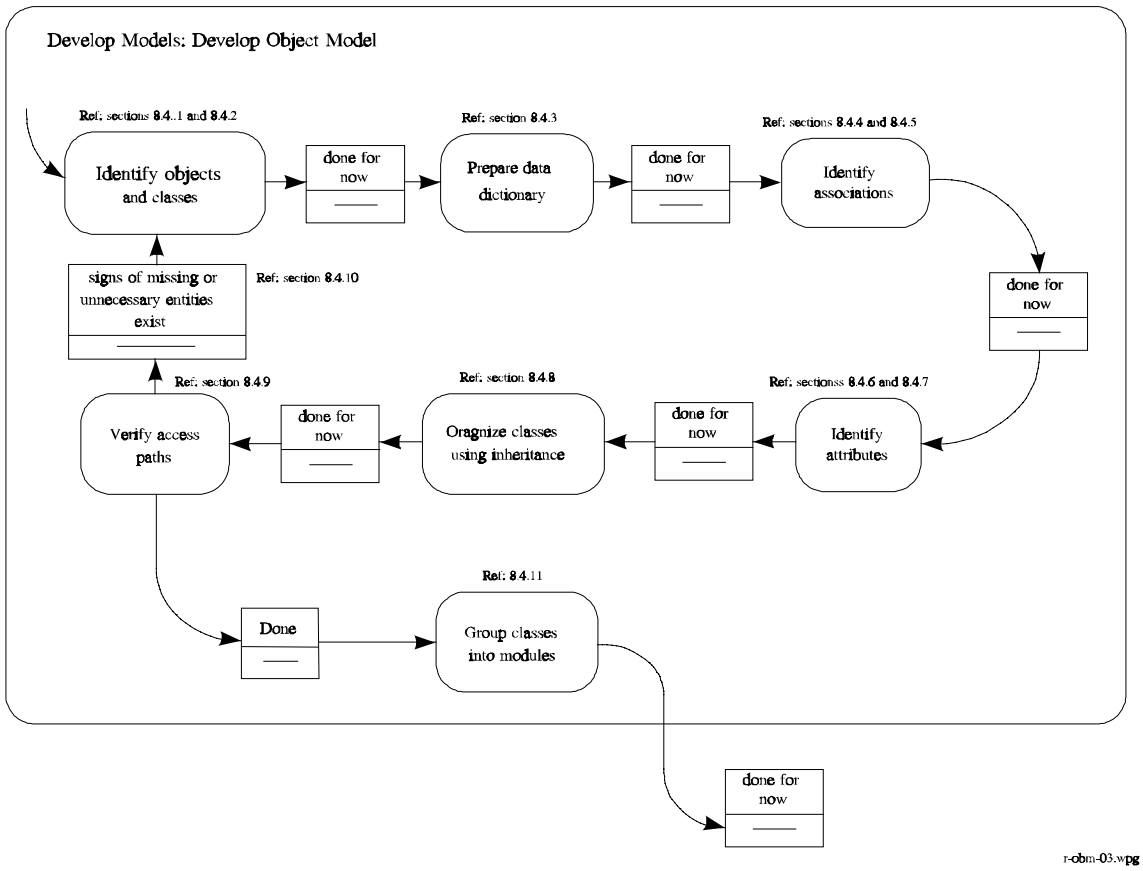


Figure 28

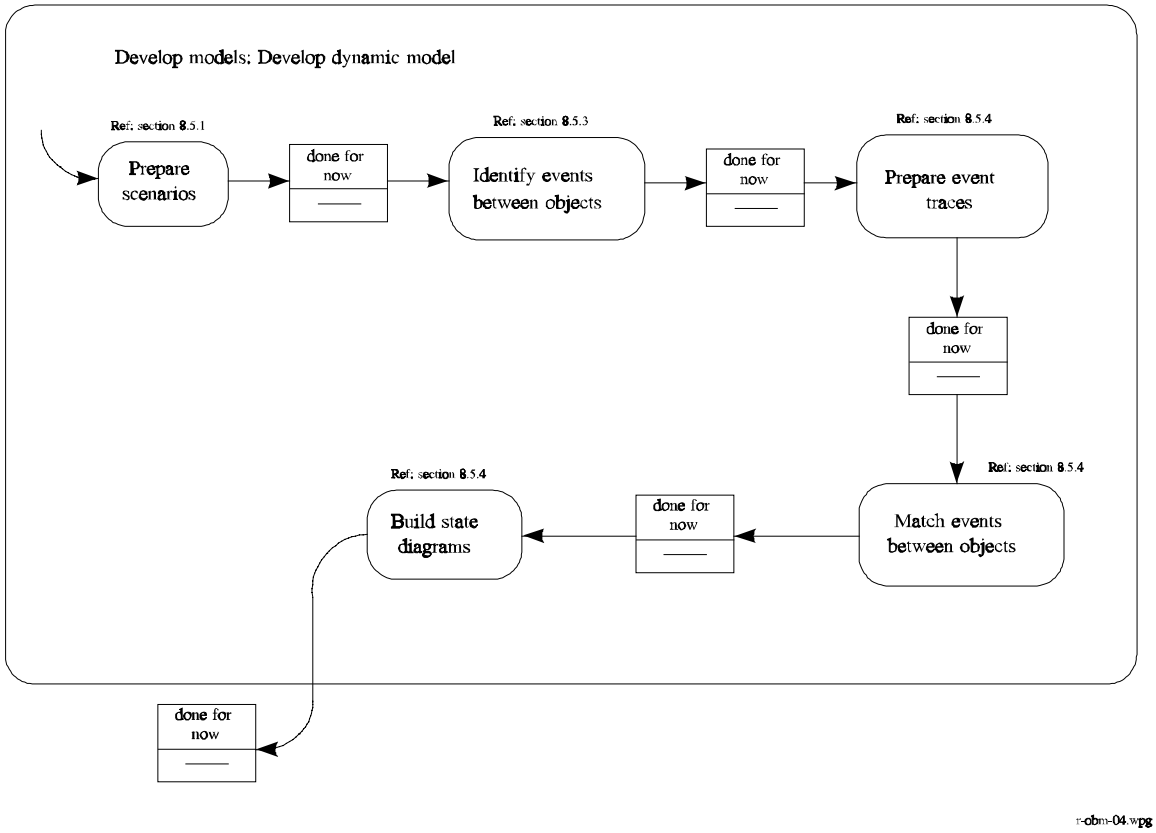
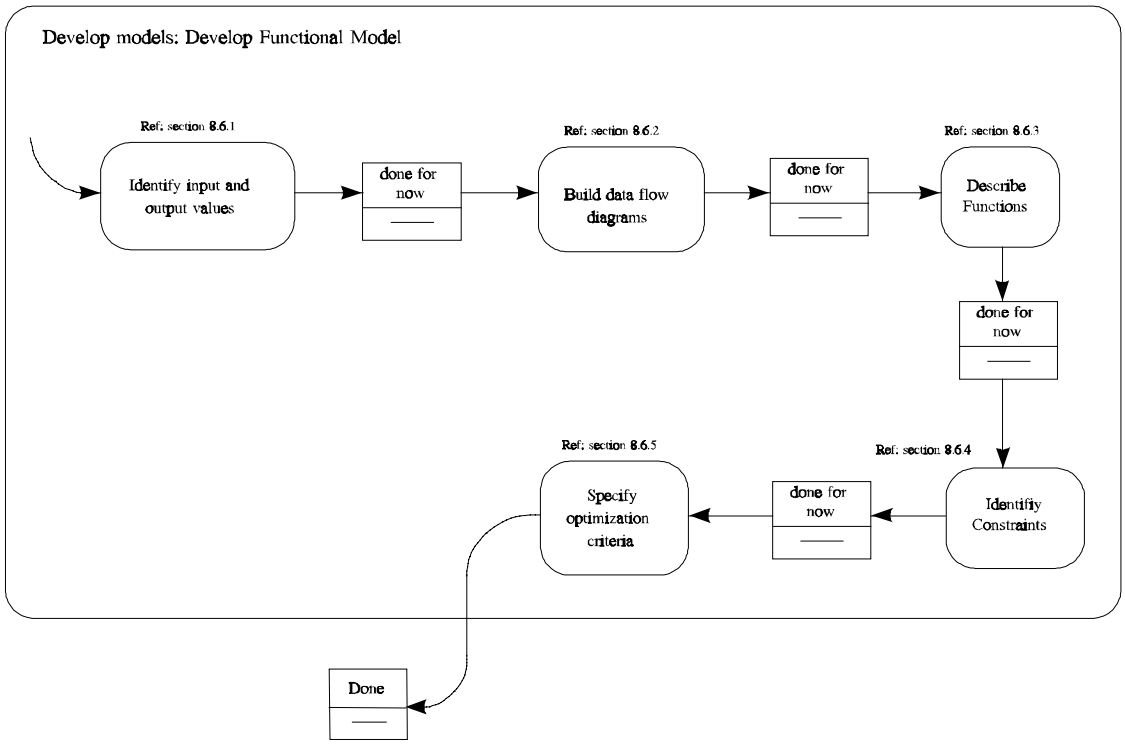
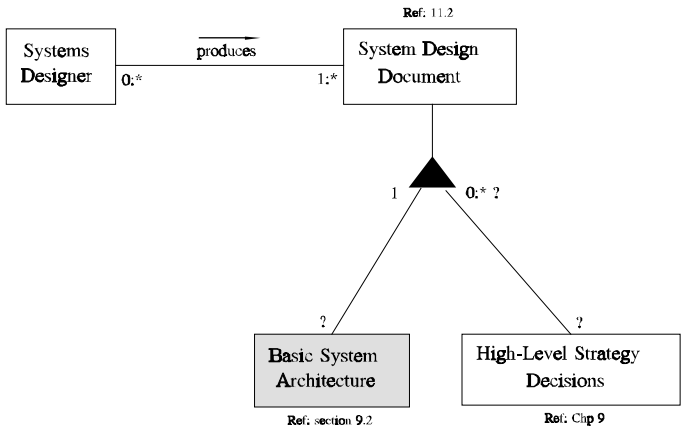


Figure 29



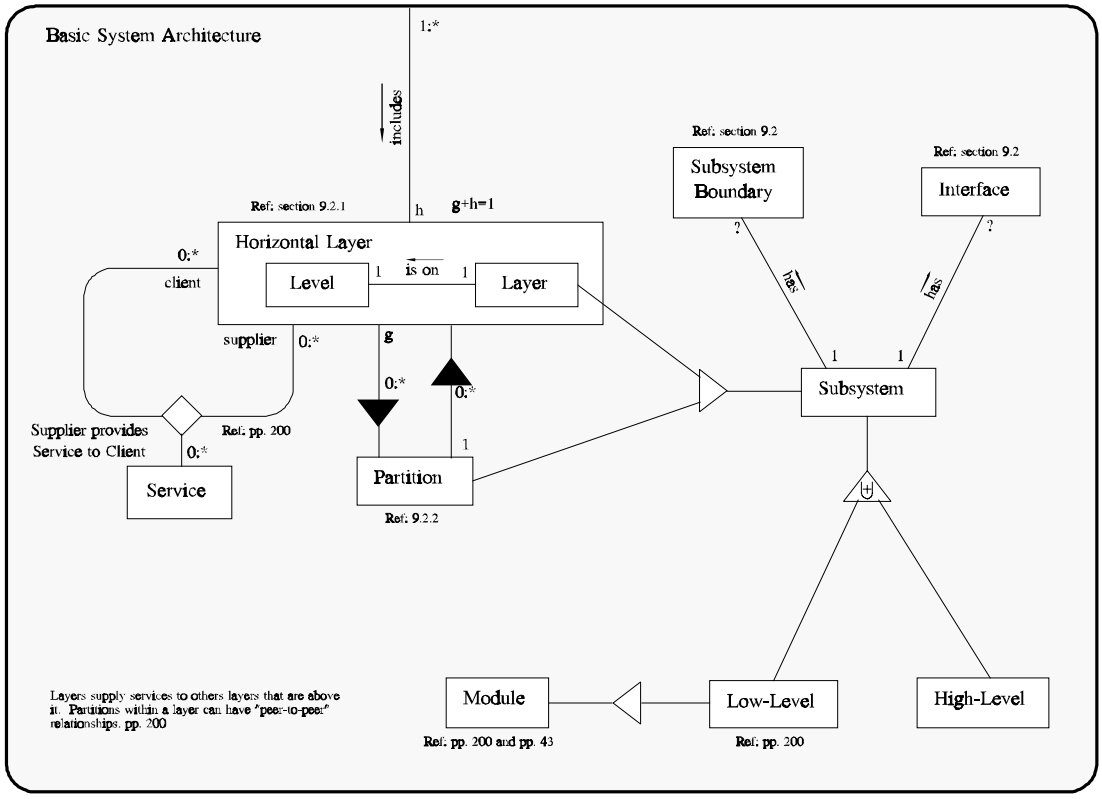
r-obm-05.wpg

Figure 30



r-orm-26.wpg

Figure 31



r-omt-27.wpg

Figure 32

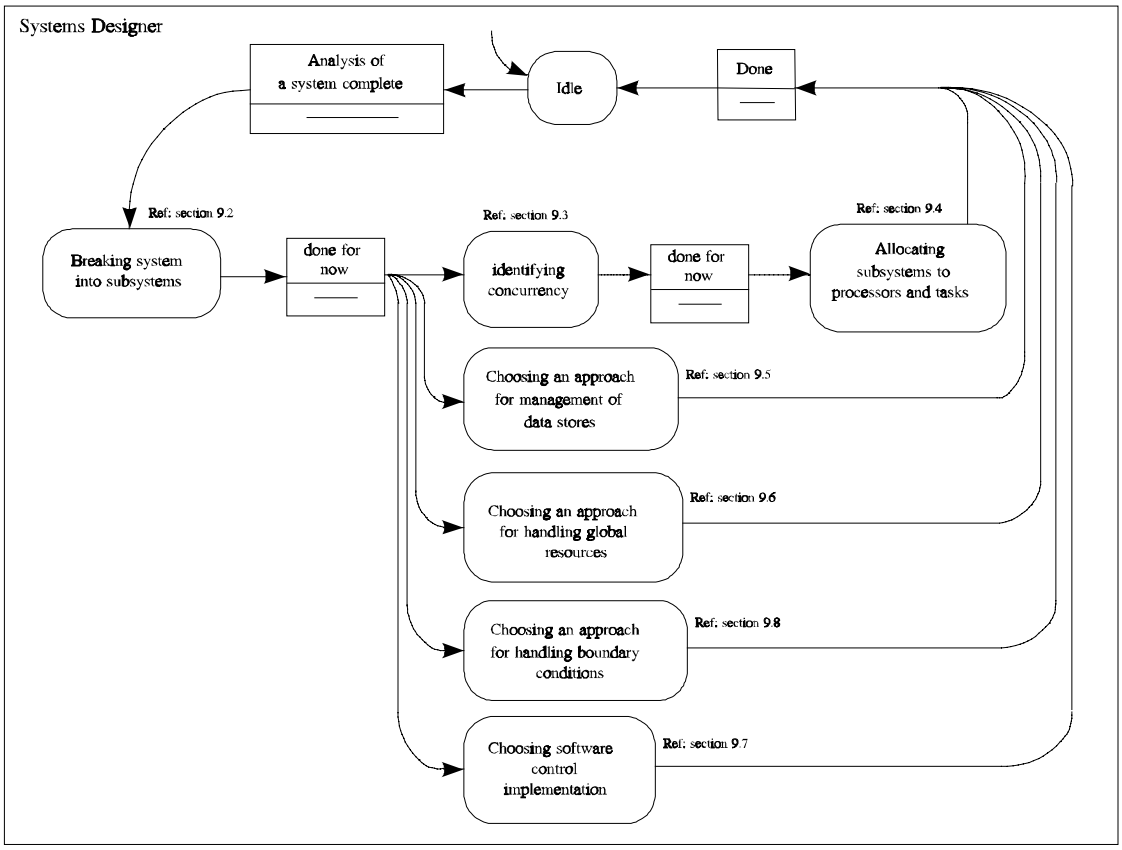
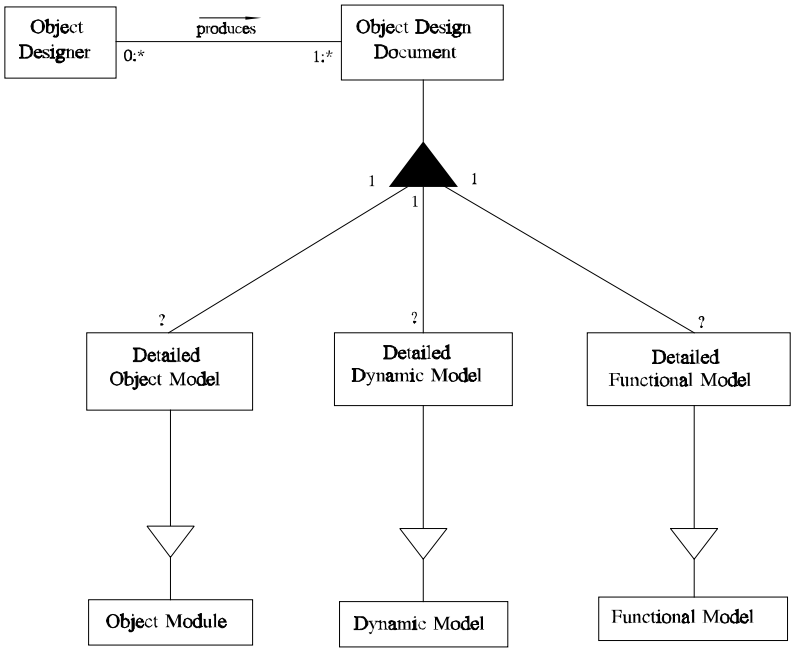


Figure 33



A Detailed Dymanic Model and Detailed Functional Model have all operations identified. Algorithms are selected for each operation.

A Detailed Object Model has appropriate data structures selected.

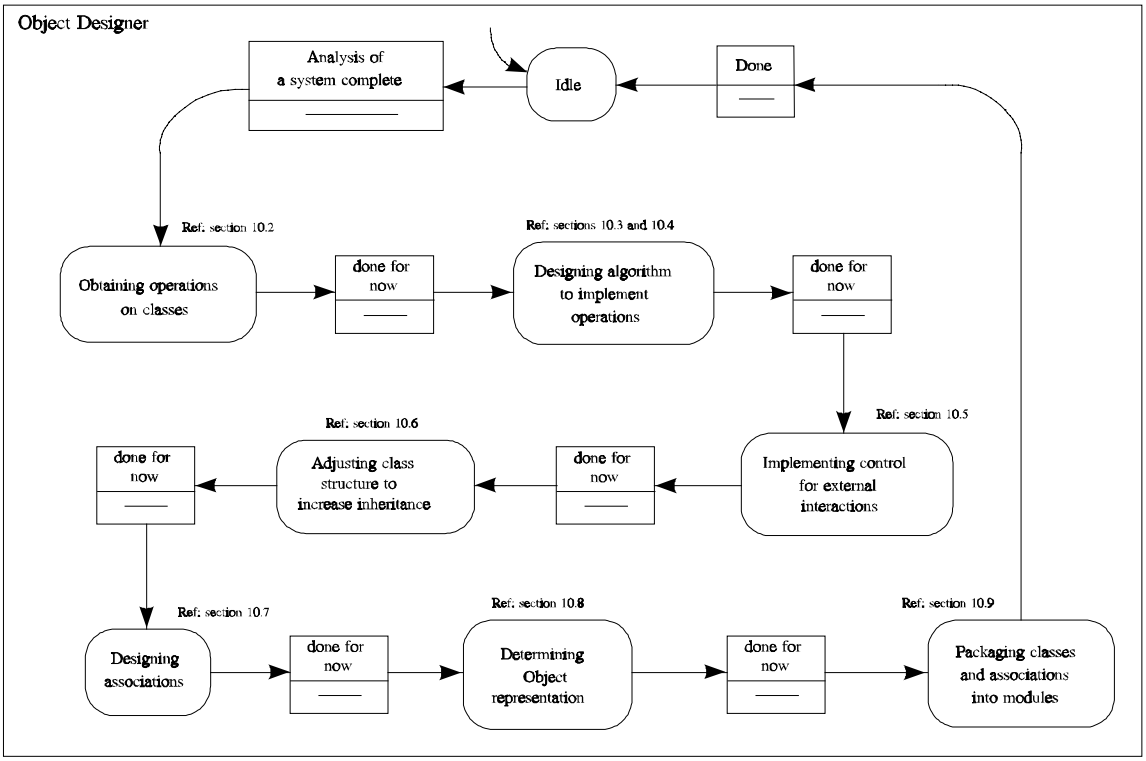
A Detailed Object Model has optimized access paths to data. Other optimizations are also in placed, such as saved derived entities and rearranging. The class structure is rearranged to increase inheritance

A Detailed Object Model has its entities packaged into object modules.

Ref. pp. 263

r-orm-28.wpg

Figure 34



r-obm-07.wpg

Figure 35