3.3.2.4 Object Communication Diagrams (OCDs)

An OCM provides a graphical representation of event communication between behavioral models and external entities. Each state model defined for an object is represented by a bubble in the OCM. External entities are represented by rectangles as before. Events specifying the interfaces among state models and interfaces between state models and external objects are represented by directed solid arcs. Events are labeled exactly the same way as they are previously labeled in the state models and their associated ADFDs.

OCMs can be structured such that objects with limited knowledge (or view) on the purpose of the system are placed in the lower part of the diagram and objects with higher knowledge on the purpose of the system are found at the upper part of the diagram For example, objects that are controlled (such as valves, switches, etc.) are placed at the lowest part of the diagram while operators (or objects which initiate events) are placed at the highest part. This convention helps in structuring the OCM in such a way that events flow from the upper parts of the diagram to the lower parts. The middle part of the OCM are made up of objects which act as agents that receive events from the upper part objects and generate events to the lower part objects.

External events generated by external objects can be one of two types as follows: solicited events are external events that were generated in response to previous activity of the system (such as a request for input event sent to external sensors); unsolicited events which can be generated any time by external objects without prior or previous action from the internal objects of the system.

An event communication pattern evolves from the above discussion as follows. System objects at the higher part of the OCM receive unsolicited events from external objects such as a human operator for example (placed at the top of the diagram). Such events would trigger or initiate a significant operation of the system as whole. Internal events are then generated and propagated downward work to lower level objects. Solicited external events might also generated at this time in response to some action taken by an internal system object such as soliciting an operator for further commands or inputs.

Figure 3.34 shows the OCM diagram for the traffic light intersection control system. The Figure specify the messages passed by the objects of classes defined in the ERD of Figure 3.31. The intersection type object passes a start_street_flashing signal to all objects of type street identified (by the street_ID) with its instance. Notice also the pedestrian class located at the top right corner of Figure 3.34 initiates a push button event to the crossing_request_button object which in turn sends a crossing_request_pending message to the lane type object (specified by the lane_ID).

3.3.3 The ICASE OOA tool Support

This section briefly describes the main features in ObjectTeam/OOA as an example of ICASE tool support for OOA. Other than providing support to develop and navigate through the OOA diagrams described in the previous section, the tool provides a checking facility to check the syntax, completeness, and consistency of the model. The tool also provides a report generation facility which generates reports summarizing the OOA model and its development

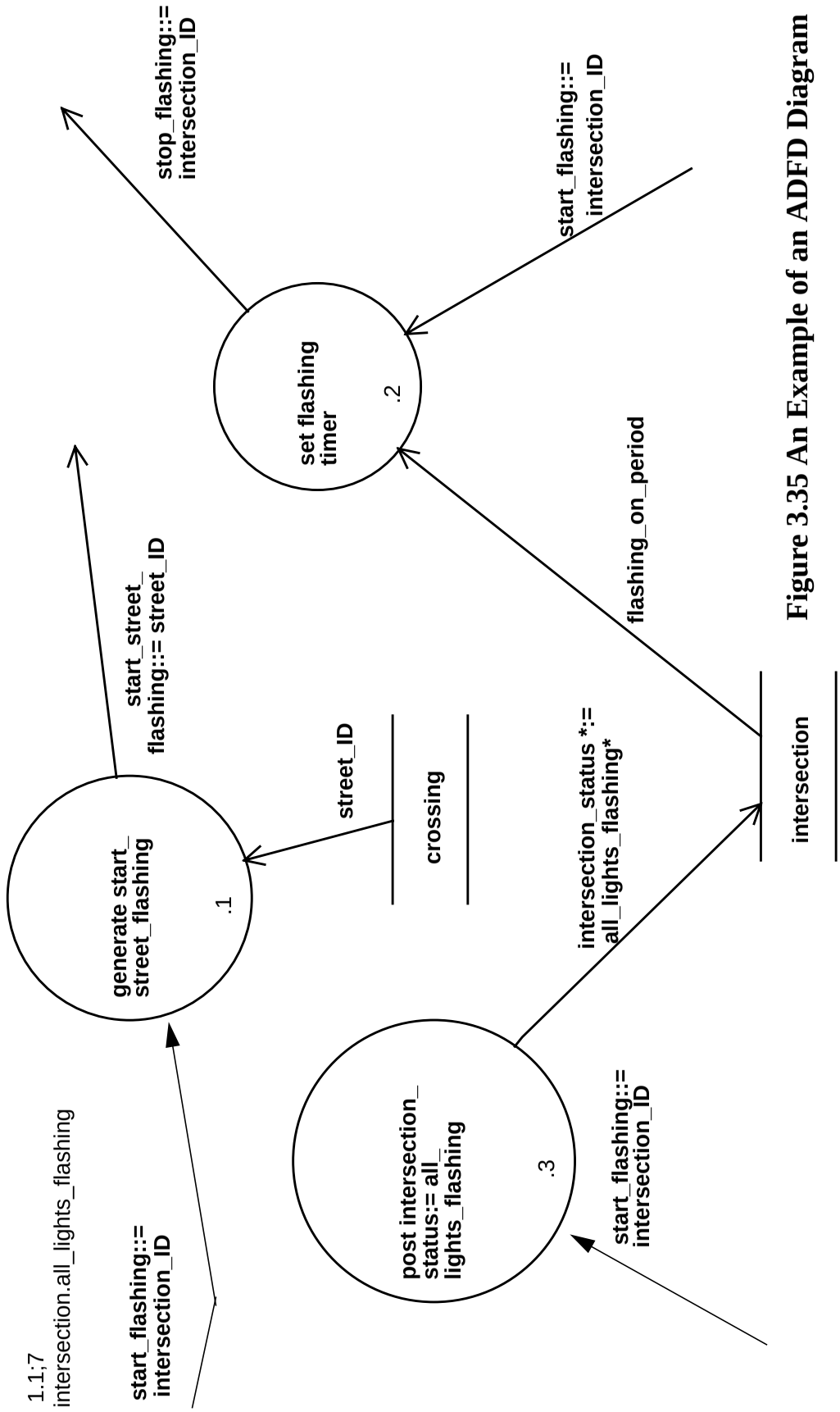


Figure 3.35 An Example of an ADFD Diagram

also create and delete object instances. The initial state of an object instance always contains an action to create the instance.

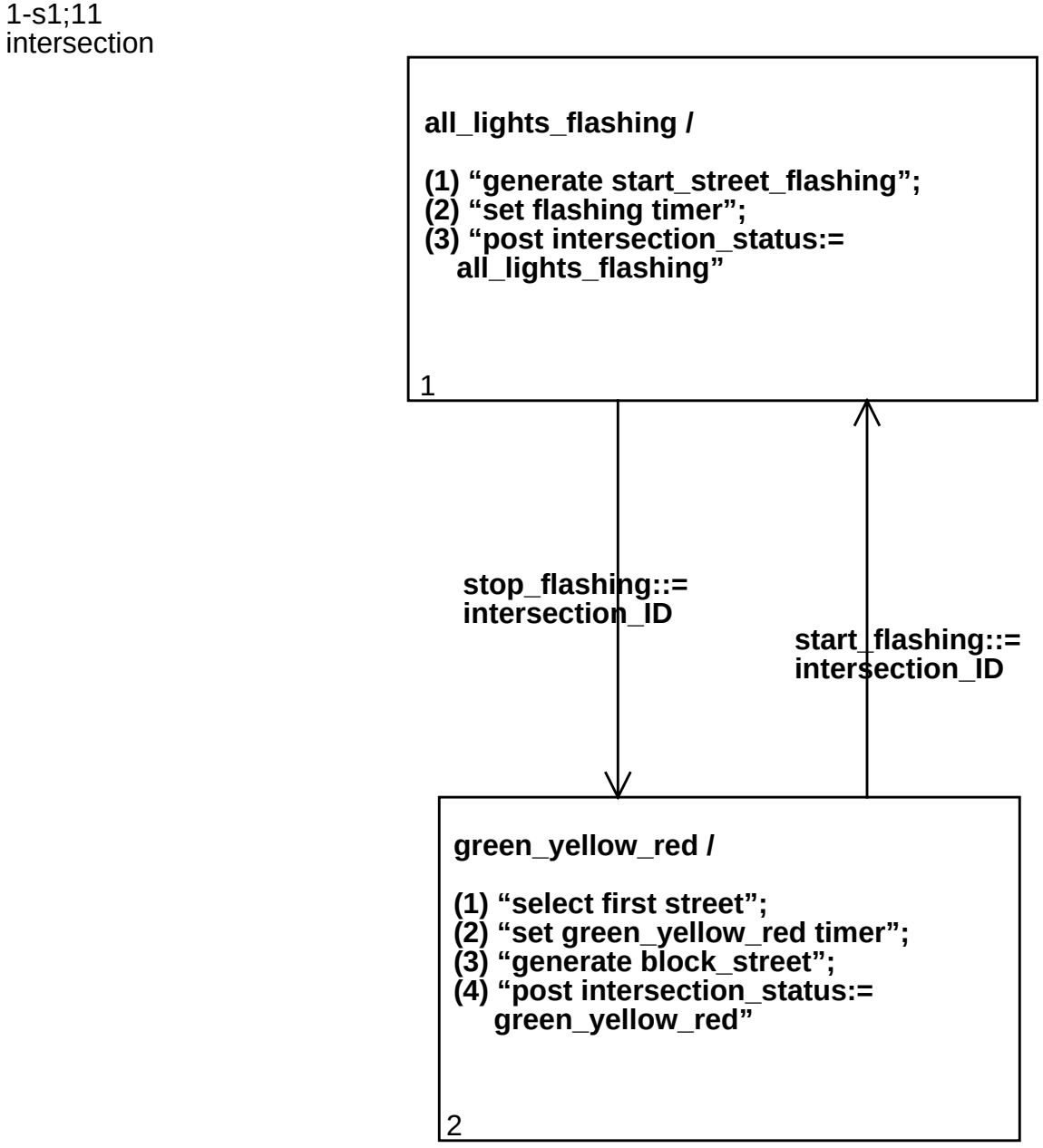
Actions are specified briefly in an action list on the text block on the state as mentioned in the state representation paragraphs above. Actions for each state are then specified in detail using Action Data Flow Diagrams (ADFDs). The notation used for ADFDs are similar to those used for the process model in the structured analysis technique described in section 3.2 with few exceptions. An ADFD consists of the following:

- 1. Processes (drawn as bubbles): represent operations such as data access from a data store; consume input event data (produced by other ADFDs) and/or input data from external entities; Produce or generate output events (to be used as input to other ADFDs) and/or output data to external entities.Processes also represent computations or data transformations.
- 2. Stores (drawn as double lines). represent a storage area for object instances. These data stores are visible by the various ADFDs specified for the various states of an object's behavioral lifecycle.
- 3. External entities (draw as rectangles). Represent objects external to the system which interacts with the internal system objects.
- 4. Data/Control flows (Drawn as solid/dotted lines).represent the flow of data and controls between processes and external entities. Generated events are shown as flows directed from a process to the empty space, and input event data are shown as input directed to a process from the empty space.

Complex processes can also be specified further (by specifying their algorithm) using process description files which are similar to the P-specs files used for describing primitive processes in structured analysis as shown previously in Teamwork/SA-RT. Processes on OOA can not be represented by lower-level ADFDs as the case, in general, in Teamwork/SA-RT. This is because OOA advocated object-oriented hierarchy and decomposition rather functional decomposition. In the former, objects are composed of other objects, or are related through an inheritance hierarchy in which case, primitive actions can be easily specified for all objects at the various levels of hierarchy.

Figure 3.33 shows an ADFD for the all_lights_flashing state of the intersection class.The diagram uses the start_flashing signal to generate the start_street_flashing event for each street in the intersection. It also sets the intersection status and the flashing timer. A stop_flashing signal is generated when the Time-out event occur. This signal is used in the STD described above in Figure 3.32 to cuase a transition to the green_yellow_red state. The diagram also generated a start_street_flashing signal for each street type object in the intersection.

Figure 3.34 Timer Object Behavioral Model



•

3.3.2.3 Actions Data Flow Diagrams (ADFDs)

Actions are the processes associated with a state and are executed or activated upon arrival to the state. These actions operate on the event data (input event data) provided by the event that caused the transition to the state. The event data contains identifier information which specify the instance of the object in which the action is executing. Actions produce data and control signals which trigger other events to occur (output event data). Actions

what happens when an instance of a given state receives a particular event. A cell may contain one of the following three entries:

- 1. The name or number (i.e., row number) of the next state for the particular current state-event combination which the cell represents. This corresponds exactly to a labeled transition in the STD.
- 2. “Event ignored” clause. This is used in case the object refuses to respond to a particular even at a given state. In this case the instance of the object stays in the same state and does not re-execute the actions specified at this state. The event however is used up by the state model in this case. The STD does not show the event ignored case at all since it only shows transitions representing recolonized and serviced events. Note that there is a difference between this case and a case in which a transition back to the same is specified. In the later case the actions specified with the state are re-executed. Event ignored simply means that the event is consumed and nothing happens.
- 3. “Can't happen” clause. is used to specify the case in which the state-event combination can never occur in reality. Since this might be a source of errors in the analysis, the analyst should provide a note explaining why (unless it is very obvious) such combination can never happen.

The last column in the STT (after the event columns) specify the actions performed at each state. Including actions in the STT is optional if they are already specified in a companion STD.

Figure 3.32 shows a simple STD for the intersection class of objects shown at the top of the ERD in Figure 3.31 The intersection traffic lights STD consists of two states, an all_lights_flashing state, and a green_yellow_red state. The set of actions for each state are listed as part of the text block on the state. The actions in the first state consist of generating an event to start the proper flashing lights signals for the each street in the intersection.(flashing red lights, or flashing yellow lights). The action “set flashing timer” starts the timer for the all_lights_flashing state.When the timer generates a Time-out signal, a stop_flashing event is triggered for the intersection. This event causes a transition to the green_yellow_red state. This state in turn sets the green_yellow_red timer which, when a Time-out event occurs, triggers the start_flashing event that causes a transition back to the first state.

State Representation

The rules governing the state descritpion in an STD (or an STT) are summarized as follows:

- 1. Give a descriptive name to the state followed by a colon.
- 2. The set of actions which take place in the state are listed as part of the text block on the state. Again each action should be given a descrptive name, these actions are specified further in Action Data Flow Diagrams associated with the state.
- 3. Certain key words signifying special actions can be specified in the text block on the state with the action list. These key words are "%generate;" and "%delete;". after the first key word a particular event is specified by a name, descriptive text, and data (if applicable). This specifies the generation of a particular event in this during this state. The second keyword “%delete;”

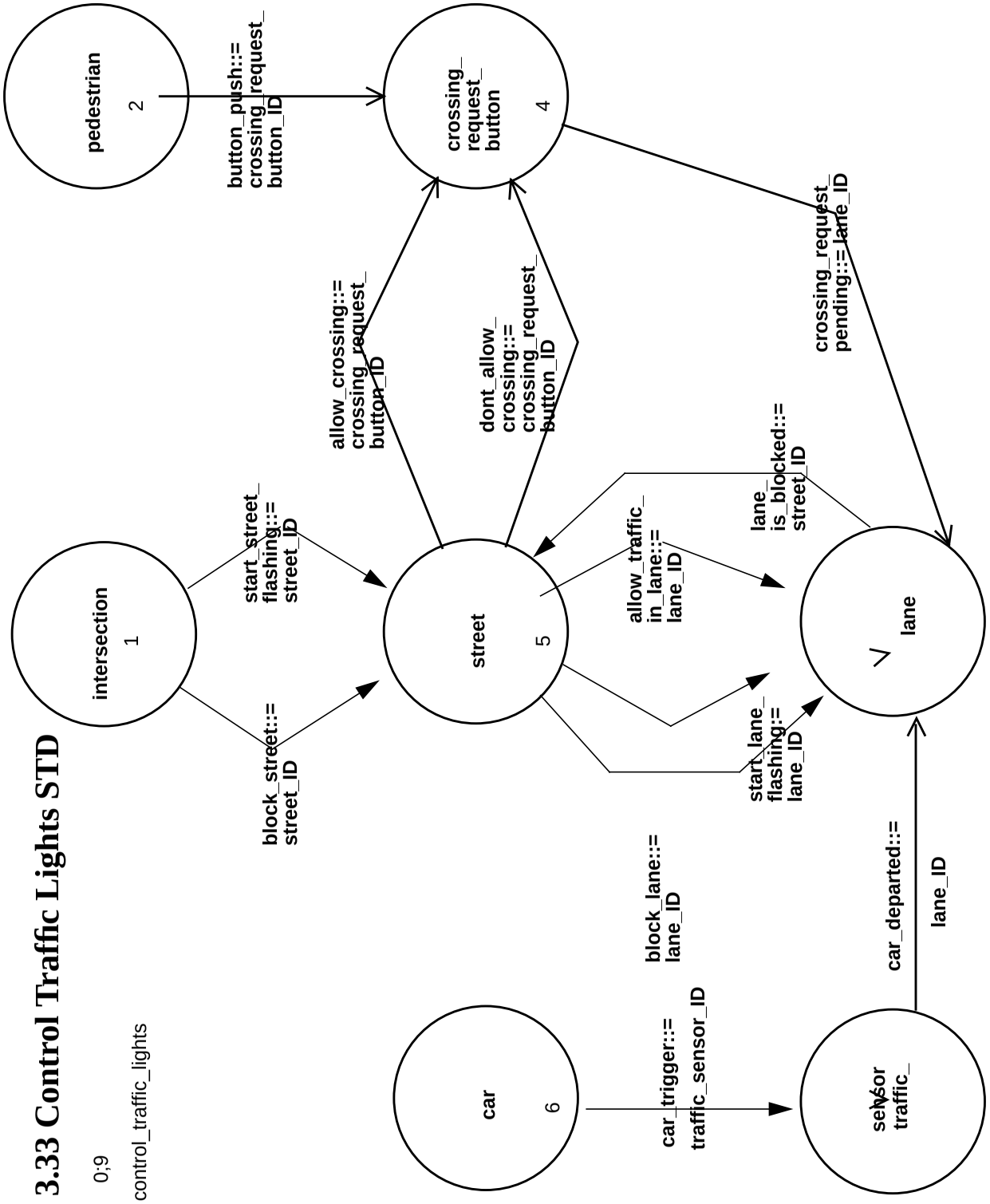
Representing Events

Transitions in an STD are caused by events. Each transition is indicated by a directed arc pointing to the direction where the state transition occurs. The event causing the transition must be specified beside the transition. Events are represented by specifying four attributes for each event as follows:

- 1. Label: a unique label must be provided for each event. This label is terminated by a colon. This label is used as an identifier for the event.
- 2. Meaning: following the event label a few words separated by underscores contain a descriptive meaning of the event. For example the words distant_object_sensed, time_out_signal, temperature_threshold_exceeded, and Reset_timer describe the meanings of different events.
- 3. Event data: since an event can be considered as a control signal, it may carry data to be supplied to the actions in the target state upon arrival to that state. The specification of the data carried by the event are enclosed between parenthesis following the event meaning text described in the 2 above. If more than one data item are specified they should be separated by commas. The data can be an identifier data such as the ID of the object where the event took place (e.g., the timer ID of the timer where the time_out_signal or the Reset_timer events occurred). This identifier data are necessary when several objects of the same type are active in the same time. The data carried by an event can also be supplemental data such the estimated distance carried by the distant_object_sensed event, or the temperature overshoot in a temperture_threshould_exceeded event.

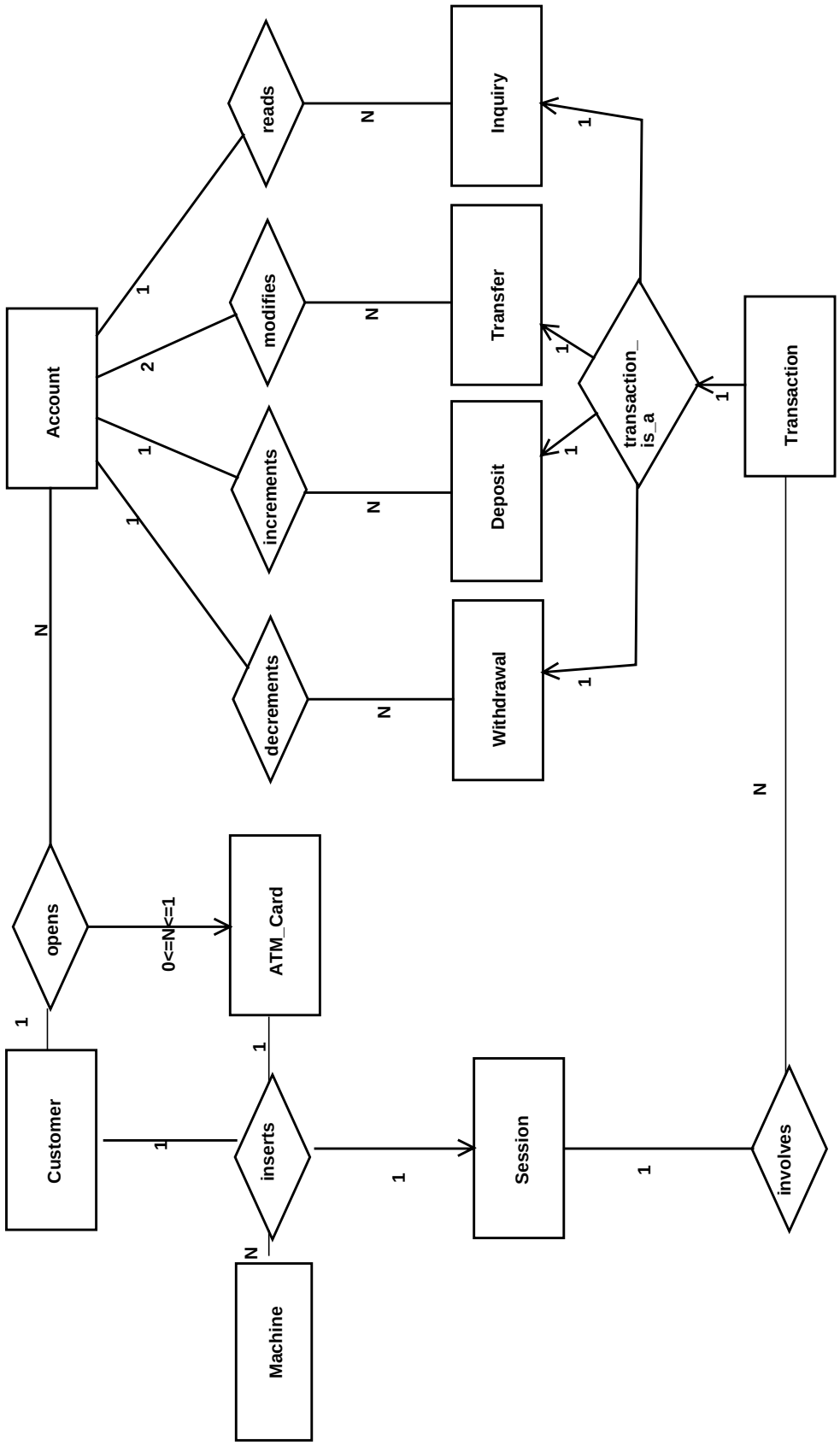
State Transition Tables (STTs)

When the behavioral model of an object is represented by an STT each row in the table represents a possible state in the state model of the object. Each column in the table represents an event that belongs to this table. The cells in the table are filled to specify


3.33 Control Traffic Lights STD
0;9
control_traffic_lights
pedestrian
2
button_push:::=
crossing_request_
button_ID
crossing_
request_
button
4
intersection
1
start_street_
flashing:::=
street_ID
allow_crossing:::=
crossing_request_
button_ID
dont_allow_
crossing:::=
crossing_request_
button_ID
crossing_request_
pending:::=lane_ID
lane_
is_blocked:::=
street_ID
street
5
allow_traffic_
in_lane:::=
lane_ID
lane
7
block_street:::=
street_ID
start_lane_
flashing:::=
lane_ID
block_lane:::=
lane_ID
car_departed:::=
lane_ID
car
6
car_trigger:::=
traffic_sensor_ID
sensor
traffic_
8

The OOA notations for the state model of an object instance consist of specifying the following:

- a) A set of all possible states of the object. A state represents a particular condition of the object instance in which a certain set of rules and physical laws applies. An instance of an object should have an initial state (also called creation state) which is depicted in an STD by a state with an incoming transition from no source.
- b) A set of events, These can be externally sensed control signals or internally generated signals which cause state transitions to occurr.
- c) A set of transitions for each state specify the new state for each possible current state-event combination, and generation of events which in turn trigger the occurence of transitions to possibly new states.
- d) A set of actions specified for each state. Each action represents an activity or an operation that must be accomplished at this state.



ATM_Network;1

Westbrook ATM Network

Figure 3.30 ERD Model of an ATM Network

relationship between class Transaction and the subtypes Withdrawal, Deposit, Transfer, and Inquiry has a 1:1 multiplicity which means that a Transaction type object must be one of these subtypes (Exclusive-Mandatory). The subtypes are also related to class Account with the multiplicity shown. Class Transfer is related to with multiplicity 2 to class Account since a Transfer type object affects the state of two objects of type account.

Figure 3.31 shows the ERD of the traffic lights intersection control example discussed in section section 3.2.3.1. The Figure shows seven classes of objects, their attributes and relationships. The intersection class has a referential attribute (preceded by an @ character) and three other attributes specifying the status of the intersection (i.e., flashing or in the green_yellow_red cycle), the flashing_on_period which specifies the time where the intersection lights are flashing, and the flashing_off period where the lights are set on the green_yellow_red cycle. Class street has a number of important attributes such as the durations for the green, yellow, and red lights, the extended duration for the green light when the number of cars waiting is beyond a given threshold (specified by another attribute), the number of seconds pedestrians should wait, and the particular flashing color and timimg for the street. Some of these attributes were mapped from the parameters specified in the requirements statements.

3.3.2.2 OOA Behavioral models

The OOA bahvioral model is defined by a set of state transition diagrams (STDs) one for each object (objects which do not have event-ordered behavior are omitted). State Transition Tables (STTs) can be used instead of or along with STDs for objects with complex behavioral specifications. Although an STT presents the same information contained in an STD in tabular form. it was mentioned by practitioners that a specification using both format are easy to follow and more readable. An STD diagramatic form is essential for understanding the lifecycle of an object. An STT representation, on the other hand, helps the analyst in verifying the completeness and consistancy of the transition rules. Filling out the STT representation forces the analyst to consider the effect of every possible state-event combination on the object behavior. It also prevents inconsistant spcifications such as specifying two differnt states as possible distinations of a state transition caused by a particular state-event combination.

The notation for STDs are similar to the notation that were described in section 3.2 for the structured analysis technique. Here however, the Moore model for STD specification is assumed. In this case, in each state in the diagram, actions performed and the signals generated accordingly are also specified along with the state description. The notation explained here which is adopted by ObjectTeam/OOA follows the notation proposed by Sally Shlaer and Steve Mellor in their second book entitled "Object lifecycles: Modeling the world in states". This second book has an updated coverage of the OOA notation supported by many CASE tools.

```
<ERD model name>.<Object name>.
```

The ERD model name is the name of the system or subsystem being analyzed. This concept helps in specifying a model for a complex system using a set of ERDs interconnected using imported objects. The concept is important since ERDs are not hierarchical as compard to Data Flow Diagrams.

The notation for specifying attributes follow the same notation used to specify data in DDEs, as discussed in section 3.21.1.4, with a slight modification to mark the naming and referntial attributes. In the case of naming (also called identifier) attributes, the charater @ is used in the begining of the attribute name. Referential attributes names are also marked by the relation idetifier connecting the object with the referred object. This identifier is specified between parenthsis after the referrential attribute's name.

Relations between objects are represented using diamond symbols. Each diamond, representing a relation between two or more objects, contains a unique relation identifier. An identifier starts with the letter R followed by a number. The connectors used to indicate the objects involved in the relationship are also labeled (for the sake of readability) with a pair of descriptive names enclosed in parenthesis that describe the relationship from the prespective of each of the participating objects. The cardinality of relations are also specified on the connectors on each side of the relation. A 1 specifies one to one, M specifies a one to many, 1c specifies a zero to one (c stands for conditional which also called conditional in the definitions given in the previous section), and Mc represents zero to many (i.e., mutliple conditional).The Figures show the three types of relations, associative, inhiritance, and aggregation.

The notation also provides for specifying referential attributes of an object. Recall that a referntial attribute associates the instances of one object to the instances of another by having an attribute in the object referring to an attribute in the other object. This is only possible if an associative relation is already defined between the two objects. The referential attribute is marked by tagging the attribute name (in the list of attributes specified for the object) with the associative relationship idetifier placed in parenthesis, and can also be specified further in the DDE defining this attribute. DDEs can be created to specify objects, attributes, and relations as will be shown in the next section.

Associative objects which depend on the existance of an associative relation between two other objects can also be specified. An associative object contains referntial attributes to identifiers from each object participating in the relatioship. An associative object is connected to the relation diamond with a connector having a single arrow head (the class Crossings in Figure 3.31 defines an associative class of objects with class intersection and class street).

Figure 3.30 shows an ERD of the ATM example discussed in section 3.2.3.2. The Figure shows the classes of objects and their relationships. The attributes of objects are not specified on the Figure. The ATM_Card class specifies associative objects which can exist for objects of type Customer and objects of type Account. The multiplicity is [0:1] . The Session class also specifies associative objects with objects of type Customer, Machine, and ATM_Card, respectively. The multiplicity gives specification such as: a Session type object might ivolve one or more objects of type Transaction, however, each Transaction type object must be associated with only one object of type Session. The inhiritance

supported by commercial CASE tools) have been standardized or declared as an industry standard. The notation supported by Teamwork is somewhat closer to the traditional structured analysis technique than the Booch notation. This is because in the former, the information model, the behavioral model, and the process model are easily identified. The Booch notation on the other hand is more powerful in specifying the object interfaces and interaction mechanism as exemplified in the scenario diagrams. In the following subsections the notating for ObjectTeam diagrams will be discussed first.

A final note on OOA notations is in order. A unified modeling language is being developed at Rational Software Corporation by three well known object methodologists. These are Booch, Rumbaugh, and Jacobson. The language is intended to provide a common simplfied architecture-centered methodology for OO analysis and design. The methodology is also intended to provide a simplified way for forward and reverse engineering, i.e., generate code, or implementation level models from design artifacts and reverse engineering some of the design back from code. The set of artifacts to be developed using the language consist of the following diagrams: a Use-case diagram, Class diagram, STD, Message-trace diagram, Object-message diagram, Process diagram, Module diagrams, and Platform diagrams.

The Use-case diagrams are used to develop use cases or scenarios that encompass a system's behavior. Message-trace diagrams, object-message diagrams, and process diagrams are used to specify the dynamic semantics of the collaborations of objects. Module diagrams are used to model the development details of the system, and platform diagrams are used to specify the physical computing topology where the software executes. Judging from the number of models mentioned, the language is more comprehensive yet also much more complex than current techniques. Efforts are under way at Rational to simplify the language and formalize (using formal logic) the metamodel which provides an unambiguous specification of the syntax and semantics of the language. For more information on this evolving technique the reader should consult the Web site of Rational at www.rational.com.

From the above discussion on the unified modeling language, it is believed that the current notation described in the following sections provide a simple introduction to the methods and techniques of OOA and prepares the ground for a better understanding of the evoloving more complex techniques.

3.3.2.1 Entity-Relationship Diagrams (ERDs)

ERDs are among the most common semantic models used for specifying data and information models. For OOA, ERDs consist of a graphical and textual notation used to specify classes of objects, attributes, and relations.

The notation for an ERD supported by ObjectTeam/OOA is described using the examples shown in Figure 3.30 and Figure 3.31. This notation is only slightly different from the Shlaer-Mellor notation. Classes of objects in the figures are specified by rectangles containing the textual name of the class. The object attributes are best described in a Data Dictionary Entery (DDE) or it can be specified in the rectangle representing the object (if there are only a few objects in the model, then larger rectangles can be used where the object name and attributes can be specified as shown in Figure 3.31). Objects imported from another ERD (i.e., an object defined in another ERD model) can be shown using the notation

implementation perspectives. While these features may aid in design, they are likely to hinder analysis. This supports the conclusion of some critics of object oriented analysis that it is more preliminary design than pure analysis and offers little real analytic support at the requirements analysis stage. The above study showed that Booch's notation has the largest number of design features while the Shlaer/Mellor notations has the least number of design features. Booch's notation in fact as described in [booch 94] contains features suited for detailed design and implementation as well.

The ObjectTeam/OOA notation is based on the Shlaer/Mellor notation which contains the following types of diagrams:

- 1. The Entity Relationship Diagram (ERD) specifies the information model that represents objects, their attributes, and relationships,
- 2. The Object communication Diagram (OCD) provides a graphical summary of the communication between objects identified in the information model.
- 3. State transition Diagrams (STDs) specify the behavioral model for each object,
- 4. Action Data Flow Diagrams (ADFDs) specify the process model that represent the data and control processing for each state of each object.

The Booch notation for OOA is based on the following diagrams:

- 1. Class Diagrams (CDs) identify the classes of objects and their relationships. This type of diagrams represents the information model as depicted by an ERD in the teamwork notation.
- 2. Scenario Diagrams: A scenario is specified by a sequence of interactions between objects and is an indication of needed object collaboration. An Object Message Diagram (OMD), or a Message Trace Diagram (MTD) is used to specify scenaios. These diagrams provide similar information specified in an OCM in the ObjectTeam notation.
- 3. A State Transition Diagram (STD) is developed for each class of objects to specify the event-ordered behavior of the class's instances similar to the procedure in ObjectTeam. STDs in the Booch Notation allow nested states in which a superstate can have serveral substates.

Examples describing the notations of the above diagrams are given in the next subsections.

The Object Modeling Technique (OMT) developed by Rumbaugh and his group is also another widely used technique for OOA. The technique is based on ERDs or Class Diagrams, STDs, and a functional model based on Data Flow Diagrams similar to the Shlaer/Mellor notation described above. However, none of the above notations for OOA (although currently

the user makes a selection first and then puts the money, most machines will still be in a state of waiting for a selection.

The example in the previous paragraph clearly shows the importance of behavior and state information when interacting objects are coordinated. The object-oriented approach is solely based on specifying and designing the system as a set if interacting objects, therefore, behavior is a critical concept which must be thoroughly analyzed, understood, and specified. Current notations for analyzing and specifying behavior will be discussed next.

3.3.2 The Current and Evolving Notations of OOA

This section focuses on introducing the current notations used for OOA models. We first discuss the notation used to specify objects, attributes, and relations. Then the notation for specifying the behavior of objects are introduced. The discussion will focus mainly on the current notation supported by ObjectTeam/OOA. The notation for OOA supported by Rational ICASE tools and presented by Grady Booch in [Booch 94] will also be briefly introduced in this section.

A large number of notations were proposed for object-oriented modeling in the past few years. Embley, Jackson, and Woodfield of Brigham Young University [EJW 95] recently compared some of the most popular notations developed for OOA. These notations are listed as follows:

- The object-oriented systems analysis by Shlaer, and Mellor [shlaer/mellor 92]
- Object-oriented analysis and design by Booch [Booch 94]
- Object Modeling Technique (OMT) by Rumbaugh [Rumbaugh 91]
- Object-oriented systems analysis by Embley, Kurtz, and Woodfield [EKW 92]
- Object-Oriented Analysis by Coad, and Yourdon [Coad/Yourdon 91]
- Object-Oriented requirements analysis and logical design by Firesmith [firesmith 93]

The study in [EJW 95] concluded that all of the above notations contain many design features (i.e., features used in the design phase) because the methods evolved from a design and

Diagram describing this example will be shown in the next section when the notation for describing objects and relations will be discussed.

OOA Models and Data Models

As mentioned in the beginning of this section, the concepts and techniques of object and relation specification in OOA are extensions of those used in data and information modeling. Data objects are defined in much the same way that objects in OOA are defined. Relations between data objects are also identified in much the same way. The major difference between a data object and an object in OOA is that the former is specified only using data structures whereas for the later a set of operations (or functions) as well as states are needed to specify the behavioral information.

The following information describes the items needed to fully specify an object in OOA.

1) Object name, a descriptive name of the object is given.
2) Attributes, each attribute should be specified by a name, its content, and data type or structure (if given in the requirements or needed to fully specify the attribute)
3) External inputs/outputs to/from the object, these are the input and output data flows and events for the object. These data flows and events include messages passed between the object and other objects as well as messages passed to/from external entities.
4) The states or modes in which the object can be in at any given time during its life time.
5) The operations or functions performed at a given state of the object dynamic behavior. Each operation is specified by a name, an interface description, processing description, performance or timing issues, and any restrictions or limitations stated in the requirements or needed to fully specify the operation.

Reiterating what was mentioned above regarding behavior and state. The behavior of an object is how the object acts and reacts to internal and external events in terms of its state changes and message passing. A good example here is the vending machine example which dispenses products. The usual behavior of such objects is that when a user puts enough money in the proper slot and then makes an appropriate selection, the selected product is dispensed. If

The cardinality (or multiplicity) of the inheritance relationship between a supertype and the subtypes is defined as follows. More examples will be given in section 3.3.2.1 when Figure 3.30 is discussed.

a) 1:1 Exclusive-Mandatory, this implies that the supertype must be any one of the subtypes, e.g., the sensor (supertype) in a specific application must be either a temperature sensor, or a pressure sensor.
b) 1:M Inclusive-Mandatory, here the supertype must be one or multiple of the subtypes. As an example in logic circuit design, a logic circuit is a supertype which must be a combination of multiple specialized logic circuits such as multiplexers, decoders, etc. Inheritance relationship with inclusion combines inheritance with the aggregation relationship which will be discussed below.
c) 0:1 or 0:M optional, an optional inheritance would mean that subtypes other than the ones specified do exist and therefore a supertype does not have to be any one of the specified subtypes. For example, a general sensor supertype (perhaps reused from the analysis of another application) is specified in such a way that it could be a supertype for other sensor subtypes not specified in this application.

3. Aggregation: The aggregation relationship defines the whole/parts relation which can be used to indicate how an object “consists of” other objects. It is used to define how a system composed of subsystems which in turn consist of several components. As an example, an object of type control panel consists of a keypad, a screen, and a switch.

Aggregation is used to define another form of hierarchy different from inheritance. The object screen in the above paragraph is a part of the object control panel but it does not have any common attributes with it. inheritance on the other hand which is defined by means of an “is a” relation, is based on the existence of common attributes. The example of a logic circuit, given above when inheritance with inclusion is defined, shows how the two concepts of defining a hierarchy of objects can exist simultaneously. An object of type logic circuit may “consist of” multiplexers and decoders. A hierarchy based on aggregation can be defined in this case to specify a particular logic circuit. However, an object of type multiplexer (or type decoder) “is a” logic circuit by itself, and a set of common attributes can be defined to specify a supertype called logic circuit. In this case a hierarchy based in inheritance also exists. A

aircraft-pilot direction, on the other hand, is 1:N which indicates that an object of type aircraft can be associated with N objects of type pilot. This gives the relation multiplicity which is not symmetrical in both directions as shown in this example.

The cardinality of relations can be 1:1, 1:N, N:N, 0:1, and 0:N. The first three are described in words as one to one, one to many, and many to many. The last two specify cases in which the relation may not exist (also called conditional or optional). In this case an object of one type is associated with zero to one (or zero to many in case of 0:N) object (s) of the other type. This is referred to as conditional relationship since the zero here indicates that the object might not be associated with the other object.

2. Inheritance: is extremely important in object oriented development. It facilitates the process of abstraction which is needed to specify generalization/specialization (or also called, supertype/subtype) relationships. Inheritance enables us to specify a hierarchy where classes of objects at a lower level of the hierarchy inherits the attributes of the classes of objects at a higher level of the hierarchy or abstraction. It significantly reduces the complexity of specifying the detailed attributes of a large number of objects who have some common attributes and behavior.

As an example of inheritance consider the relationship between several types of sensors in a given system (such as temperature sensor, pressure sensor, motion sensor, smoke sensor, etc.). A class of objects of type sensor can be used to define an inheritance relationship. The attribute of objects of type sensor could be the ID of the subsystem or the part of the system in which the sensors are located, and the value of the data sensed. The subtypes should have their own unique attributes which could be their own ID and a particular preprocessing operation or function (e.g., filtering of noise) defined on the raw data they sense.

- Referential attributes: These attributes exist only if the instance of an object is tied to instances of other objects. In this case information referring to the other objects are included. Examples are: an instance of an aircraft contains information referring to the ID numbers of pilots flying the aircraft; a magnetic tape instance contains information on specific tapes drives which can be used to mount the tape, etc.

To develop a meaningful set of attributes for each object the analyst must examine the requirements of the system and specify all possible attributes that belong to the object according to the types of attributes specified above.

Relationships among objects

Objects contribute to the behavior of a system by interacting or collaborating with one another. Also objects may consist of other objects and may inherit the same attributes from a more abstract object who in turn may have many siblings. The main types of relationships outlined by Booch in [Booch94] are summarized as follows:

1. Association: denotes a semantic dependency between two objects. It does not specify the direction of this dependency, nor does it indicate the exact way one object relates to another. However, text can be inserted along with the definition of the relation to imply these semantics by specifying the role each object plays in relationship with the other. These semantics are sufficient during analysis since what is needed is to identify such dependencies and their cardinality or multiplicity as will be discussed below.

As an Example of an association relationship between two objects consider the relationship between the objects pilot and aircraft. The relationship identified as "a pilot can fly one aircraft" and by "an aircraft is flown by one or more pilots" is a bidirectional associative relationship between the two objects.

The cardinality of the pilot-aircraft direction of the above relation is 1:1 which states that an object of type pilot can be associated with one aircraft at any given time. The cardinality of the

actions such as searching for the tape, finding an available tape drive, then mounting the tape on the tape drive.

Grady Booch [Booch94] defines an object as follows:

> "An object has state, behavior, and identity; the structure and behavior of common objects are defined in their common class. The state of an object represents the cumulative results of its behavior. It encompasses both the static and the dynamic properties of the object. The behavior is how an object acts and reacts in terms of its state changes and message passing. The identity of an object is that property which distinguishes it from other objects."

To distinguish clearly what should not be considered an object, it is important to examine the name or properties identifying the object. In general, an object should never have an operation or procedural name. For example, in software involving matrix manipulations, matrix inversion should not be considered as an object since the name is characterized by the inversion clause which is a procedure or operation implemented on matrices. On the other hand, the tape mount object mentioned above is considered as an object since it is characterized by a unique number (the tape mount number), a particular tape, and an available tape drive. This is similar to a drivers license which is issued with a unique number, for a particular driver to use an available car.

Object Attributes

For each object selected for inclusion in the analysis model, it is important to specify the set of attributes which define the object and clearly distinguish its conceptual boundaries from other objects. These attributes can be divided into three types as follows:

- Naming attributes. These attributes provide information on the arbitrary labels or names attached to each instance of an object. Examples are ID numbers, title number, license number, names, or code names, etc.
- Descriptive attributes. These attributes give descriptive information intrinsic to each instance of an object. Examples are: altitude, longitude, and latitude information for aircraft; Capacity and size of a magnetic tape, etc.

The notation for Data and information models and for OOA will be discussed in the next section. This section aims at discussing the conceptual basis of OOA models and its relation to data modeling. We first discuss the concepts of an object, attributes of an object, and relationships between object. Then we discuss the relationship between these concepts and the concepts of data modeling. Finally, the section ends with a discussion on the behavior of objects and its specification.

Identifying Objects

In examining the statements of the requirements, the set of objects needed for specifying the system can be identified as follows:

- External entities providing inputs or consuming outputs to or from the system under development. Examples of such entities in real time systems are sensors, actuators, CRTs, operators, control panels, etc.
- Entities that are part of the information domain of the system such as reports, records, displays, windows, and signals (e.g., sensor readings, output controls, operator commands, communication massages, etc.).
- Entities which establish the context of the problem. For example in a robot mounted magnetic tape system, the robot, the robot controller, the tape drive, the tape, the tape storage are all entities which establish the context of the problem.

The Occurrences of events which happen during the system operations such as the completion of a sequence of actions can also be viewed as objects. As an example in the system consisting of robot mounted magnetic tapes on tape drives, a tape mount is considered to be an object. A tape mount is an object which is created by a user request and the completion of a sequence of

3.3 Object-Oriented Analysis (OOA) Using ICASE

The OOA technique introduced briefly in section 3.1.2 is discussed in detail in this section. The section begins with an introduction to the object-oriented modeling methodology. The current and evolving OOA notations are discussed. ICASE tool support provided in ObjectTeam/OOA is then described with examples.

3.3.1 Introduction to Object-Oriented Analysis and Data Modeling

Structured analysis as described in the previous section is based on the concepts of functional decomposition which led to the structured programming principles. Similarly OOA is based on the concepts of object-oriented programming (OOP). As mentioned in section 3.1.2, the OOP approach evolved from simulation programming in which the entities or objects of a simulated system are created and destroyed at run time. These objects are specified by a name and a set of attributes. Code is developed to simulate or implement the identified behavior of these objects.

The OOA model is an extension of the concepts found in data modeling. Data and information modeling has been used for some time in applications in which data and the relationships that govern that data are complex such as in database applications. Data modeling terminology and graphic notations are very similar to those used for OOA to define the objects, their attributes and their relationships. The Entity-Relationship Diagram (ERD) used for data and information modeling is also used in OOA as the top level diagram. The other lower diagrams in OOA specify in detail the behavior of each object and the dynamics of interactions between objects. The ERD however is very important since it specifies the system as a set of interrelated objects.