Developing Shlaer-Mellor Models Using UML

Stephen J. Mellor
Neil Lang

Project Technology, Inc.
10940 Bigge Street
San Leandro, California 94577
(510) 567-0255
http://www.projtech.com

This position paper describes how to represent Shlaer-Mellor Object-Oriented Analysis work products using the Unified Modeling Language. The Unified Modeling Language (UML) is a language, a notation, for expressing models of object-oriented designs. It does not prescribe a software development process, i.e., a set of techniques, guidelines and steps for constructing a set of related work products.

The Shlaer-Mellor Method [SM1] is a well-defined and disciplined software construction process based on the explicit separation of the analysis of the application and the specification of the software design. In this method the system is divided into distinct, separate subject matters called domains, and each domain is analyzed using Shlaer-Mellor Object-Oriented Analysis. The method prescribes a set of well-defined analysis work products. The software design for the system as a whole is specified in a separate domain called the Software Architecture.

Given a sufficiently-powerful modeling language, the Shlaer-Mellor Object-Oriented Analysis work products may be expressed in any notation. Because the UML is a powerful and flexible notation, it is possible to follow the Shlaer-Mellor process and express the Shlaer-Mellor concepts and work products using UML.

This document provides a mapping from the Shlaer-Mellor work products into the UML notation, so that a project can use the Shlaer-Mellor concepts and process and yet be UML-compliant. There is no change to the process of developing Shlaer-Mellor models; the work is done in the same order and the same work products are produced.

The UML notation used here is that defined in the *UML Notation Guide,* version 1.0 of 13 January 1997. All section references that appear in this technical note refer to the *Notation Guide*.

We proceed by taking each of the Shlaer-Mellor work products in turn, and showing how to represent that work product using UML. Note that the UML is designed to express requirements analysis, design, and implementation concepts. Because Shlaer-Mellor OOA models capture analysis concepts exclusively, only a subset of the UML notation is required to represent the work products. Consequently each UML notational element does not always have a Shlaer-Mellor equivalent.

1. Domain Chart

UML defines a *package* (§ 2.7) to be a grouping of elements; it is commonly used to group classes together. The UML defines a *dependency* (§ 4.25) between packages as a semantic relationship between the model elements in the packages. The UML shows a package as a rectangle with a tab, and a dependency as a dashed arrow pointing to the "depended on" package.

Represent each domain as a package. Place the name of the domain within the rectangle representing the package. Do not show the contents of any of the packages.

Represent a bridge as a dependency pointing from the client domain to the server domain. Do not label or annotate the dependency.

Figure 1.1a shows a Shlaer-Mellor notation Domain Chart, and Figure 1.1b shows the corresponding Domain Chart using UML notation.

Discussion: The UML does not have the concept of domain separation—indeed, the *Notation Guide* itself mixes the definition of the language with both presentation and notation.

The UML definition of a package as a group of related model elements more closely fits the Shlaer-Mellor definition of a subsystem. But then a domain is also a collection of closely related subsystems, so it is not unreasonable to use the UML package concept for a domain.

Similarly, the UML does not have the concept of a Shlaer-Mellor bridge in the sense of a client domain making use of the facilities of a server. Rather, the UML defines dependency in terms of "a change to the target element may require a change to the source element in the dependency." Of course, this is precisely the type of dependency that a Shlaer-Mellor bridge externalizes so that the link between two domains can be changed without changing *either* domain.

The semantic content of a Shlaer-Mellor domain chart is quite low—it's the thinking about the subject matters that's important. That thinking can easily be represented using UML. But do be careful to distinguish a domain from a subsystem.

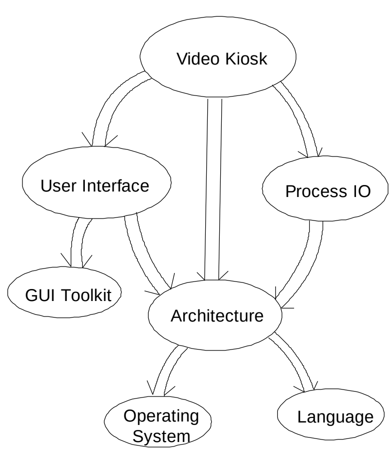

Figure 1.1a: Domain Chart for Kiosk System

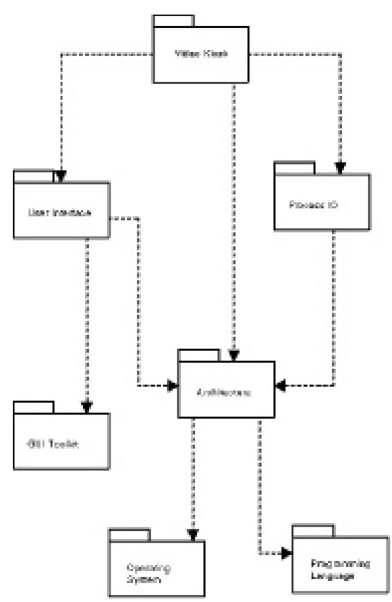

Figure 1.1b: Same Domain Chart in UML

2. Object Information Model

2.1 Objects and Attributes

A *Class Diagram* in UML captures the classes, their internal structure, and the relationships between them (§ 4.3). Like a Shlaer-Mellor Object Information Model (OIM), the Class Diagram defines only static aspects and does not describe the dynamic behavior.

In UML, a *class* has three compartments: a *name compartment* (§ 4.4) at the top, a middle compartment to hold a list of *attributes* (§ 4.14), and a lower compartment to hold a list of operations. We use the name compartment for the object name, number and keyletter. We place the attributes in the middle compartment, and we leave the lower compartment empty because a Shlaer-Mellor OIM does not show processes assigned to objects.

Represent each object in the OIM as a UML class. Place the name in the top compartment, centered, boldfaced, and titled[1] to conform to the UML style guidelines. Show the object number and keyletter (in this order) in braces below the object name.[2]

Represent the attributes of an object as a list of UML attributes in the middle compartment. Add the domain-specific data type of the attribute using the UML syntax (§ 4.14):

visibility name : type-expression = initial-value { property-string }

We examine each of these syntactic elements in turn.

The UML defines *visibility* to be one of *public*, *protected* or *private*. The UML recommends that a tool should have the capability to suppress the visibility, though it should not be undefined. The Shlaer-Mellor process defers defining the visibility of an attribute until the software architecture is available. Therefore, tag each attribute as *public* and suppress the visibility marker. (See discussion below.)

The *name* is the name of the attribute.

The *type-expression* in UML is a language-dependent specification of the implementation type of an attribute. The language we choose for the expression of a Shlaer-Mellor work product is Shlaer-Mellor OOA in which each attribute has a domain-specific data type such as *temperature* or *pressure* that we later map to an implementation-language data type. Domain-specific data types supersede the attribute domain definitions described in *Object-Oriented Systems Analysis: Modeling the World in Data* [SM2] and are described further in Appendix B. For the UML *type-expression* use the domain-specific data type defined for each attribute in the OIM.

The *initial-value* is a default value for the attribute, which you should include if appropriate. (The definition of the domain-specific data type includes the specification of a default value.)

In UML, a *property string* (§ 3.5) comprises a set of comma-delimited *property specifications* enclosed in braces '{' and '}'. Each property specification has the form:

keyword = value

where the keyword is the name of a property and value is an arbitrary string denoting its value. If the keyword stands alone, it denotes that the property indicated by the keyword is true. We used a property string for the object keyletter and number above by implicitly defining the property specification keywords for a given model to include all the analyst-specified object numbers and keyletters. Therefore an analyst need only set to true the keywords corresponding to a given object's number and keyletter.

---

[1] i.e. each word begins with an upper case letter

[2] This construct is actually a property list, described in more detail below.

There is nothing in UML that corresponds explicitly to identifying or referential attributes. Again we model these concepts using property strings. Use the property specification keywords I and Rn (the relationship number) to indicate identifying and referential attributes. Always use the short form { I } or { Rn } in which the Boolean value is taken to be true rather than the longer form { I = true }. Use the keywords I2, I3 for secondary and tertiary identifiers if desired. Similarly use the property specification keyword M to indicate a computed or mathematically-dependent attribute.

The following table lists some kinds of attributes and their combinations:

Kind of Attribute	**Shlaer-Mellor Notation Example**	**Property String in UML**
Primary Identifying Attribute	*	{ I }
Secondary Identifying Attribute	*2	{ I2 }
Attribute in multiple identifiers	*1, *2	{ I, I2 }
Referential attribute	R3	{ R3 }
Referential identifying attribute	* .... (R4)	{ I, R4 }
Mathematically dependent attribute	(M)	{ M }

The OIM in figures 2.1a and 2.1b show objects in Shlaer-Mellor and UML notation respectively.

Discussion. Shlaer-Mellor defers defining the visibility of an attribute in the model of a domain. The developer tags each attribute to be one of the options defined to be available in the software architecture during the definition of the bridges. Within a single domain then, Shlaer-Mellor does not require that the analyst specify visibility. On the other hand, the UML requires its definition but allows suppression of the visibility marker. Consequently, it makes no difference which visibility marker you select.

Because the purpose of analysis is to expose information, we recommend defining the visibility to be *public*. Alternatively, if you wish to avoid the mistaken impression that the presence of an attribute on a OIM necessarily implies violation of encapsulation, you could define the visibility to be *private*.

2.2 Binary Relationships

A *binary association* (§ 4.17) in UML captures a combination of analysis and design concepts. A binary association is shown as a solid path connecting the class symbols, named with a single verb phrase. A variety of adornments may be shown at each end of the line.

Represent a Shlaer-Mellor binary relationship as a UML binary association. Name the relationship with the concatenation of the relationship number and a single verb phrase, and place this combination near the middle of the association line. Choose the single verb phrase from one of the two verb phrases normally used to name the relationship and use a solid triangle at the left or right end of the verb phrase to indicate the direction in which to read the name. For verb phrases located next to a vertically-drawn association line use a solid triangle pointing to the right to indicate a direction of top to bottom.

UML reserves each end of the association line for the *association role* (§ 4.18), with the adornments:

- multiplicity (§ 4.19),
- aggregation indicator,
- qualifier, (§ 4.20),
- role name,

- ordering,
- navigability.

We discuss each in turn.

In UML, *multiplicity* is a combination of multiplicity and conditionality in Shlaer-Mellor. It is an analysis concept. Use the following correspondences:

Shlaer-Mellor Concept	SM Symbol	UML Symbol
One Unconditional (x:1)	—>	1
One Conditional (x:1c)	—>c	0..1
Many Unconditional (x:M)	—>>	1..*
Many Conditional (x:Mc)	—>>c	0..*

Do not use arrowheads since they represent how to link the classes in design (navigability).

In UML, the *aggregation indicator* is used to indicate aggregation or composition. If the diamond at one end of an association path is hollow, then the implication is that it is an aggregation. If the diamond is filled, then it's composition (§ 4.23). Shlaer-Mellor thinks of aggregation as a design concept and hence there is no corresponding concept in the OIM to map to either type of aggregation.

In UML, the *qualifier* is an attribute or set of attributes whose values partition the set of related objects across an association. Since the referential attributes formalizing a relationship in a Shlaer-Mellor OIM serve the same purpose, do not add a qualifier to the model.

In UML, the *role name* that can appear at each end of an association path reflects the role of the class in the association. Hence in a relationship Dog Owner OWNS/IS OWNED by Dog, the role name for the Dog object would be IS OWNED BY. Or, because UML names roles using nouns, more likely it would be OWNER. Unfortunately, this is quite the opposite of the Shlaer-Mellor convention where we associate the verb phrase with the object of the sentence, and the verb phrase associated with Dog would be OWNS.

In the Shlaer-Mellor Method, roles are typically captured in the name of the participating objects and there is no need for this adornment. Consequently, we recommend that you *not* use the role names, despite their convenient location.

Both *ordering* and *navigability* are design-relevant constructs and hence are not required in the analysis models.

The OIM in figures 2.1a and 2.1b show relationships in Shlaer-Mellor and UML notation respectively.

2.3 Associative Objects

An *association class* (§ 4.21) in UML is an association that also has class properties. UML shows an association class as a class connected by a dashed line to an association path. The UML concept corresponds closely to the Shlaer-Mellor concept of an associative object.

Represent an associative object with an association class connected to the corresponding association. Treat the associative object like any other object in listing the attributes, etc. To indicate a M-M:M relationship, add M to the property string in the name compartment.

The OIM in figures 2.1a and 2.1b show an associative object and the corresponding association class.

Discussion: UML treats the association and its corresponding class as a single model element with a single name. In a Shlaer-Mellor OOA model the name of the associative object is always a noun and

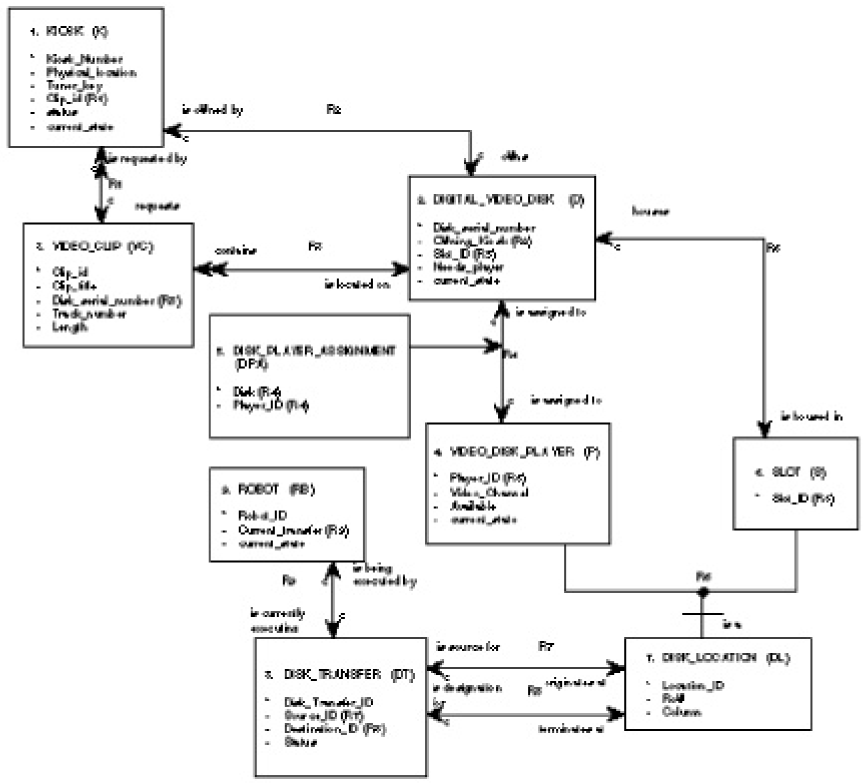

Figure 2.1a: Object Information Model for Kiosk System

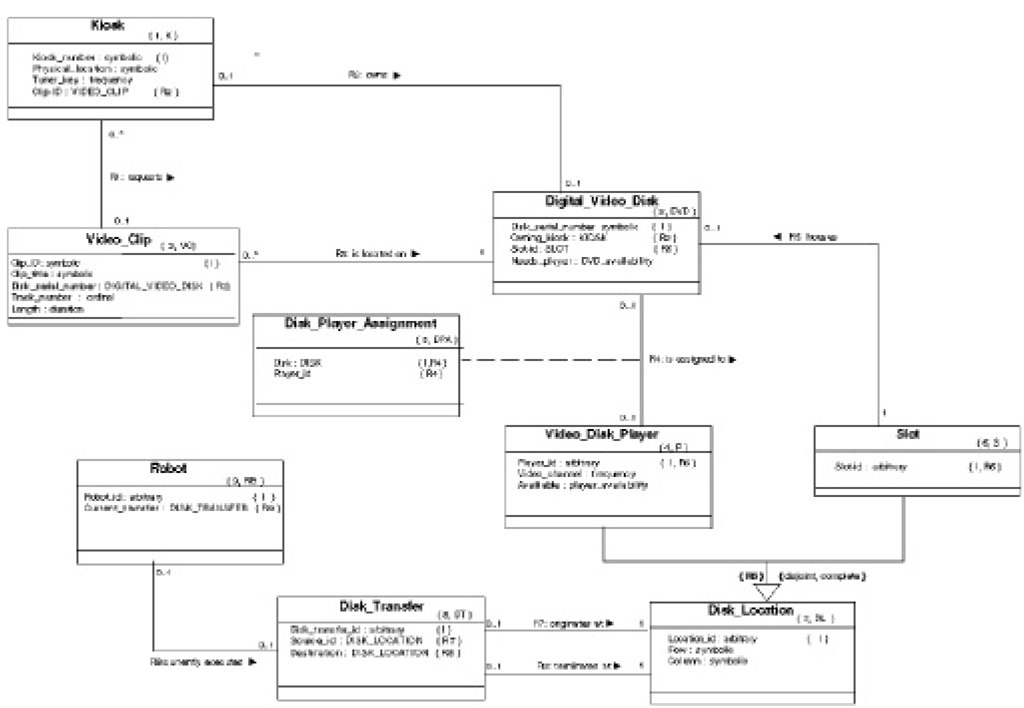

Figure 2.1b: Object Information Model in UML

the name of the relationship it formalizes is always a pair of verb phrases. To be consistent with UML we should formally treat the single name of the class and the association as the concatenation of the Shlaer-Mellor OOA object name and one of the verb phrases. Then show in the association class only the object part of the full name and on the association line only the verb phrase part.

2.4 Subtype/Supertype Relationships

Generalization (§ 4.24) in UML is a relationship between a more general element and a more specific element that adds additional detail. UML shows generalization as a solid line from the more specific element to the more general element with a large hollow triangle at the more general element end. We apply the generalization construct to classes to model Shlaer-Mellor subtypes/supertypes.

Represent a subtype/supertype construct with a generalization line directed from each subtype to the supertype. For maximum overlap with current notation, always show a generalization structure as a tree with a shared segment pointing to the class for the supertype object and branches to each of the classes for the subtype objects. Add a property string with the relationship number next to the shared segment of the generalization structure. Do not add a *"discriminator: powertype"* label to the model.

UML provides a set of predefined constraints to indicate some semantic constraints among the classes. The UML semantics in the absence of explicit constraints are *incomplete* (there may be further subtypes beyond those specified) and *disjoint* (an instance of a supertype belongs to one and only one subtype). To represent the full Shlaer-Mellor semantics of partitioning into mutually exclusive subsets, annotate the shared segment with {complete, disjoint}.

Discussion: The semantics of a Shlaer-Mellor subtype/supertype relationship are tightly specified. Instances of the supertype object are partitioned completely and mutually-exclusively into instances of the subtypes. In contrast, UML classes and subclasses are related much more loosely. For instance it is not required to define or show all subclasses derived from a class. Since the constraint set {complete, disjoint} corresponding to the Shlaer-Mellor semantics differs from the default, we show the constraints on the model. Alternatively one could omit disjoint and simply show a {complete} constraint.

3. State Models

UML models behavior using a variation of Harel's Statecharts [H1]. In UML, actions can be specified for transitions, upon entry into states, upon exit from states, and even within states. UML also offers a continuously executing activity within a state that can be interrupted upon receipt of an event. In addition, UML allows for a superstate with states embedded within it.

The Shlaer-Mellor Method prescribes a much more concise state modeling formalism [SM2] derived from Moore [M1] with flat, deterministic state models where (uninterruptable) actions are executed upon *entry* into states. In the Shlaer-Mellor Method we describe complex behavior by abstracting additional states and objects, each of which is at the same granularity.

In UML, a *state* (§ 8.2) is shown as a rounded rectangle with up to three compartments—one each for a name, state variables, and internal activity.

Represent a Shlaer-Mellor state in UML as a UML state with only a name and internal activity compartment. Place the name of the state in the name compartment, combined with the state number, if desired. In the activity compartment, write "entry /" to signify that the action is to be executed on entry to the state. Do not create "do" or "exit" actions. Enter the action either as an informal action description or as formal action language, just as it normally appears in a Shlaer-Mellor state model (UML does not specify a syntax for action descriptions.).

UML defines events very broadly: conditions, signals, invocations and the expiration of some passage of time all qualify as events. Shlaer-Mellor uses only signal events. In UML, a signal *event* (§ 8.4) is defined by:

event-name '(' comma-separated-parameter-list ')'

where each *parameter* in the comma-separated-parameter-list is:

parameter-name ':' type-expression

Concatenate the event label and event meaning to form the UML-event name. Use the name and domain-specific data type of each Shlaer-Mellor event data item to create each parameter. Do not use a semicolon to separate the identifying attributes from the supplemental data. For example:

Shlaer-Mellor Notation	**UML Notation**
K1:Request Video Clip (Kiosk Number; Video Clip Title)	K1_Request_Video_Clip (Kiosk Number : symbolic, Video Clip Title : symbolic)

UML shows a *transition* (§ 8.5) as a solid arrow pointing from the starting state to the ending state. Each transition carries a transition string that has the following general format:

event-signature '['guard-condition ']' '/' action '^' send clause

where the event-signature describes an event with its actual arguments.

Because Shlaer-Mellor labels a transition only with the event name and a list of parameters, do not specify any guard conditions, actions, or send clauses. Use only the event-signature part of the transition string to annotate the transition.

Represent a transition into a creation state as a transition from a UML-initial state to the actual creation state. Label the transition with the creation event (A UML-initial state is a pseudo state shown as a solid filled circle.).

Represent a final state in UML just in the same way as the Shlaer-Mellor notation: a state with no transitions out of the state. UML includes the concept of a final (pseudo) state but it is associated with a transition from a state to the enclosing superstate. Hence this type of final state is not used in representing Shlaer-Mellor models.

Although UML permits labeling a transition with a time expression to indicate a transition occurs at that (absolute) time or after a specified time interval, continue to use the Shlaer-Mellor delayed event formalism to represent such time-related events.

There is no direct analog in UML for the State Transition Table. Nevertheless continue to build a STT (either manually or using table features supported by your tool) for each active object to specify its behavior completely.

Figures 3.1 (a) and 3.1 (b) show the corresponding Shlaer-Mellor and UML state models.

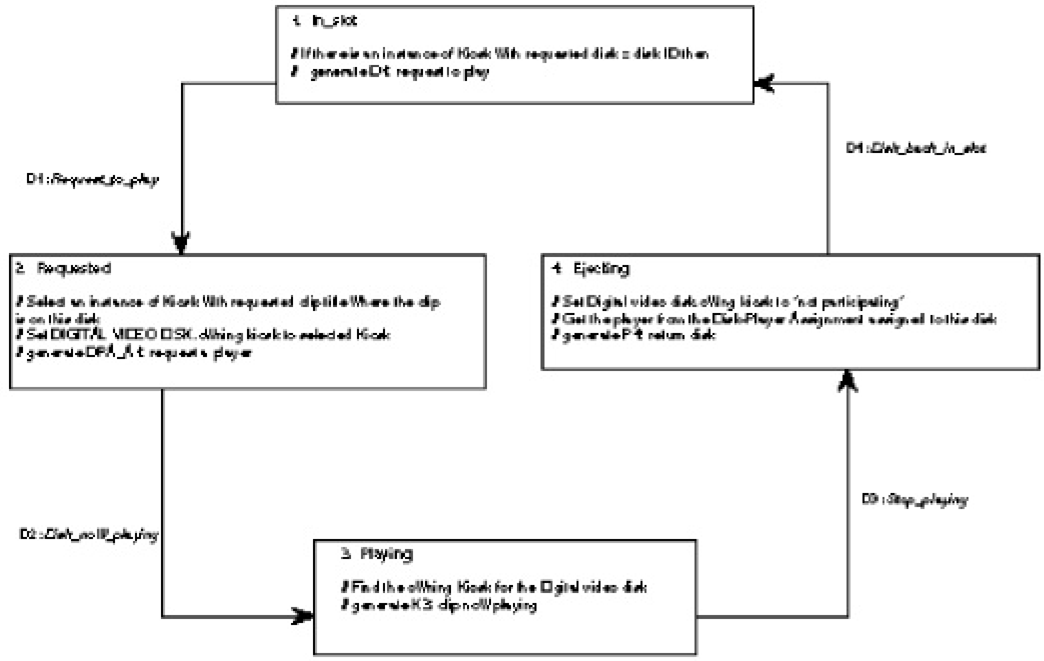

Figure 3.1a: State Model for Digital Video Disk

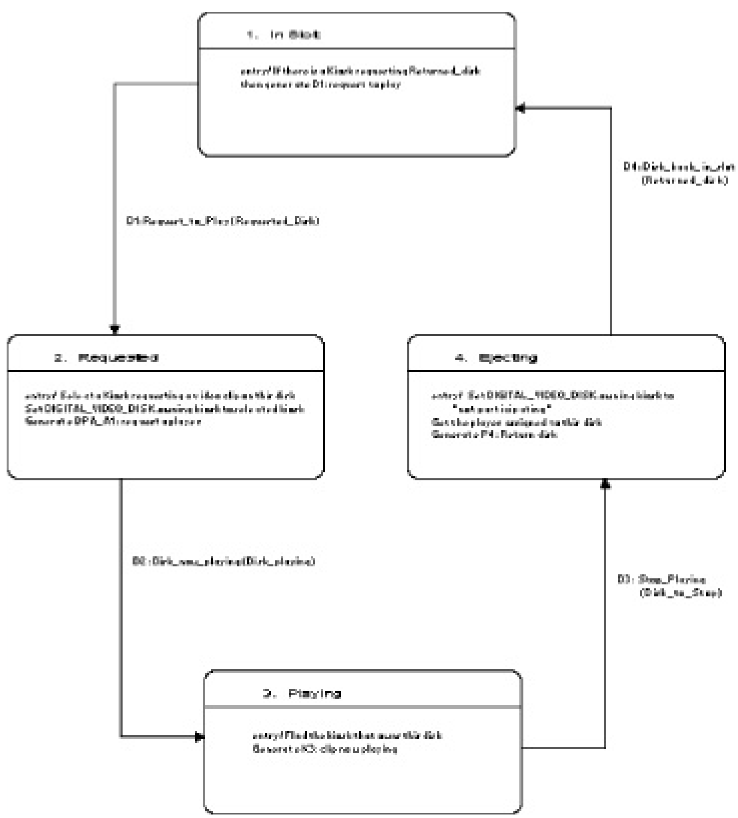

Figure 3.1b: State Model in UML

4. Object Communication Model

UML defines a variety of diagrams to show communication at the (UML) object level. We use one of these diagrams—the collaboration diagram—to represent the Object Communication Model.

In UML, a *collaboration diagram* (§ 7.5) is a network of (UML) objects and links between them. UML shows each object on the collaboration diagram as a rectangle with an upper name compartment and a lower attribute compartment. The top compartment shows the name of the object followed by its class all underlined to indicate the object-level nature. The lower compartment shows the attributes of the object and their values.

A *link* (§ 7.9) is an instance of an association between objects and is shown as a path between them. Each link is annotated with a *message flow* (§ 7.10)—an arrow labeled with a message label—to indicate the sending of a message from one object to the other. The link is used to transport the message to the target object. A message may be implemented in many different ways: direct procedure calls, sending of signals, the direct raising of an event, etc. UML specifies various kinds of arrowheads to denote different kinds of messages, but since these are implementation decisions they have no place in an analysis model.

To produce an OCM represent each OOA object as a UML object. In the top compartment place the name of the OOA object preceded by a semicolon. Underline the object name to follow the UML object convention. (See discussion below) Do not add the attribute compartment.

Represent an event generated in one OOA object to another as a link connecting the two rectangles. Do not place an arrowhead at the destination end of the link.

Place a message flow, i.e., a labeled arrow next to each of the links. To indicate the direction of the message flow use a half-stick arrowhead (the UML notation to indicate asynchronous flow of control).

UML allows for the specification of multiple items (sequence numbers, predecessor sequences, return values) in a message label in addition to the message name itself. Simply label the message with the name of the generated event. Do not include the event data.

Figures 4.1a and 4.1b show the corresponding Shlaer-Mellor and UML Object Communication Models.

Discussion: Note that the definition of an object in Shlaer-Mellor refers to a typical, but unspecified instance of some abstraction, whereas an object in UML is a specific instance of a class. Consequently, a Shlaer-Mellor Object Communication Model shows communication between typical instances, while a UML collaboration diagram shows communication between specific instances of classes.

UML distinguishes an object from a class by separating the name of an instance of the class from the class name by a colon. UML also uses underlining to indicate the object nature of a diagram. Hence, UML represents the (Shlaer-Mellor) instance of Kiosk named Amtrak Station as Amtrak Station : Kiosk. UML permits the use the shortened form : <class name> (a colon, followed by the class name, all underlined) to refer to an unspecified instance, hence : Kiosk refers to an unspecified instance of Kiosk. This is the form that we use in building an OCM.

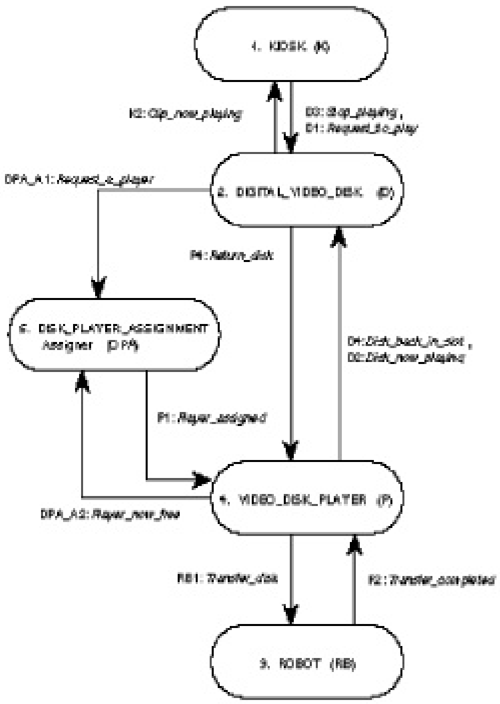

Figure 4.1a: Object Communication Model for Kiosk System

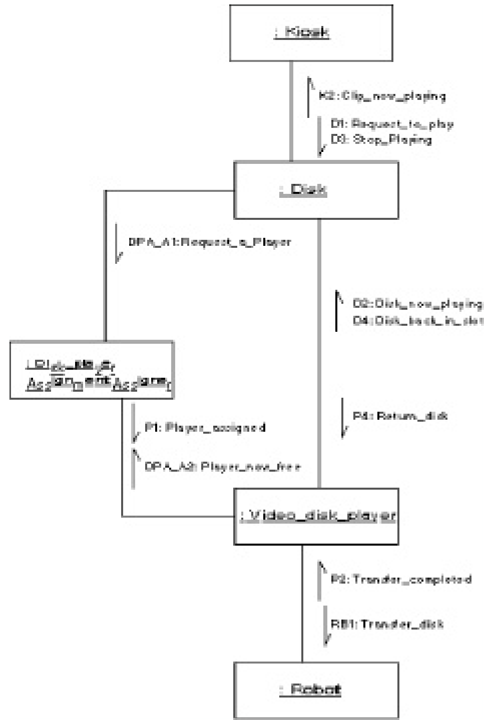

Figure 4.1b: Object Communication Model in UML

5. Thread of Control Chart

UML defines another diagram—the sequence diagram—to show a pattern of interaction among the (UML) objects arranged in time sequence. In UML, *a sequence diagram* (§ 6.1) shows the objects participating in an interaction as they evolve over time and the message communication between them. One axis (usually the vertical one) represents time. The objects are placed along the other axis. A vertical dashed line represents the *object lifelin*e (§ 6.2) of each object. A lifeline captures the *activations* (§ 6.3) of its objects, i.e., periods of time in which the object is executing some activity. Activations are denoted by a thin rectangle along the lifeline; the top and bottom of the rectangle indicate the initiation and completion time respectively of the activation.

Messages (§ 6.4) represent communications between objects and are shown as horizontal solid arrows pointing from the sender object's lifeline to that of the destination object. The arrow is labeled with the name of the message and its arguments and optionally sequence numbers and guard conditions. UML also permits variations on the type of arrows to further specify the nature of the message passing mechanism (e.g., procedure calls versus asynchronous messages).

Because of the great similarity between the thread of control chart and the sequence diagram, we use the sequence diagram to represent the Thread of Control Chart. Each instance in a Shlaer-Mellor thread of control is represented by an object on the sequence diagram and each event appears as a message.

Represent each instance in the thread with a UML object placed at either the top of the sequence chart or where that instance comes into existence (if it is created during the evolution of the thread). Label the instances as follows:

value-of-primary-identifier : object-name

Underline the labels to indicate the UML object nature of these entities in the diagram. If an instance is deleted during the thread, show that on the diagram by terminating its lifeline with a large X.

Show each event in the thread of control chart as a UML message by drawing a solid horizontal arrow from the generator of the event to the receiving instance at the appropriate time. Self-directed events are shown as arrows that loop back to the originator's lifeline. Label each message with name of the event and (optionally) the values of the event data passed.

Represent the action executed by an instance (upon completion of a transition) as a short activation, i.e., a thin rectangle along the lifeline. Label the activation using a UML note with the name of the state. Show any events generated during the action as messages originating during the activation.

Discussion: Shlaer-Mellor Thread of Control Charts often capture a large number of instances, states, and events. In many cases it is practical to only number the states and to label the events with only the event label omitting the full event name and actual values of the event data. Accordingly, when drawing the sequence diagram, label the activations only with the number of the state, and label the messages with only the event label when space is a consideration.

Note that UML allows for a variety of arrowheads on messages to distinguish among asynchronous and various synchronous messages. The Shlaer-Mellor process again defers this implementation decision to the architecture. Consequently use a filled arrowhead and do not attempt to further elaborate the message passing mechanism.

Figures 5.1a and 5.1b show a Shlaer-Mellor Thread of Control Chart and the corresponding UML Sequence Diagram.

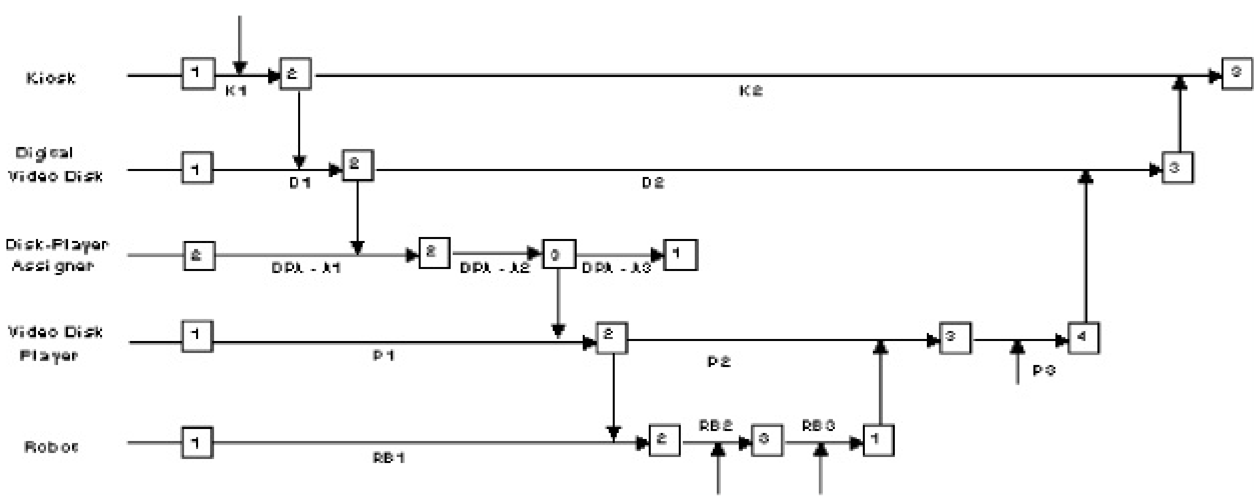

Figure 5.1a: Example of a Thread of Control Chart

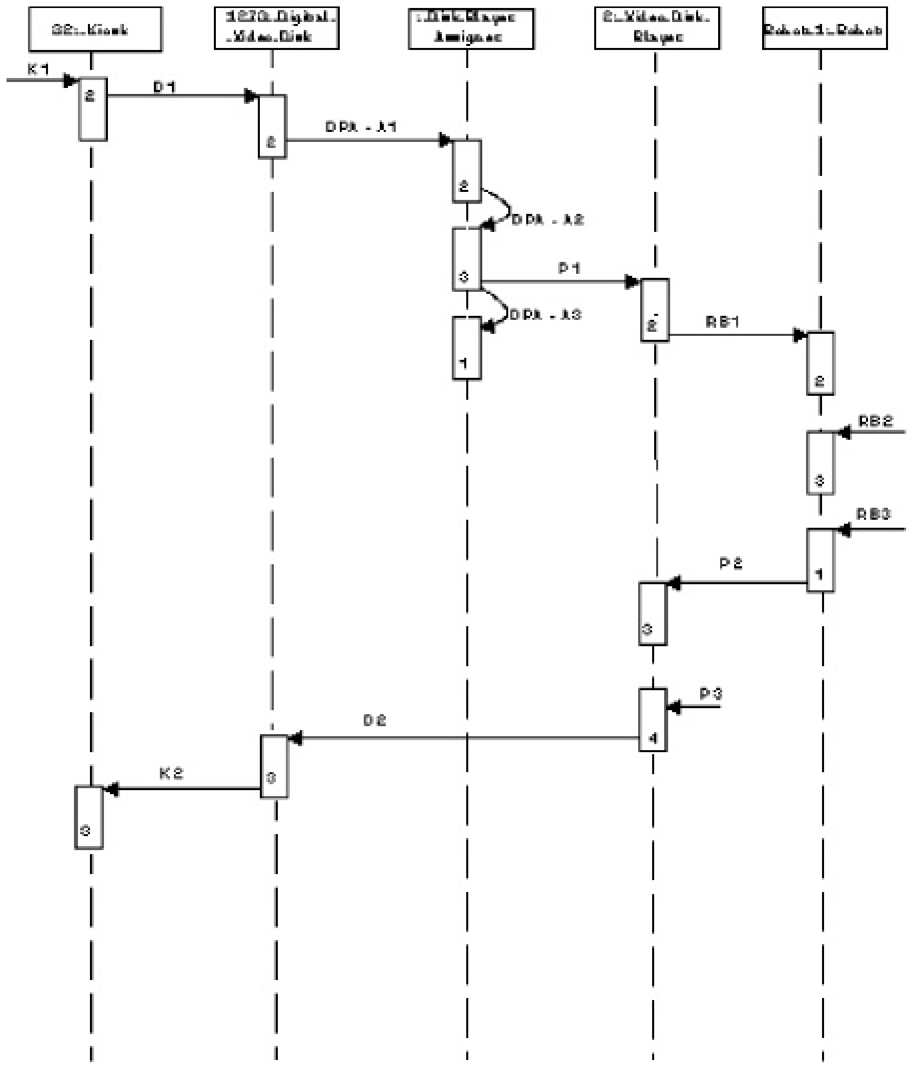

Figure 5.1b: Equivalent UML Sequence Diagram for the Thread of Control in 5.1a

6. Process Models

UML does not define a formal action language for process models. Hence continue to develop process models using the current action language.

7. Summary

The Shlaer-Mellor Method defines a process and a notation for representing domain knowledge in a formal manner. We defined the Shlaer-Mellor notation to capture pertinent domain knowledge without any predisposition to a particular design style—though the partitioning principles of Shlaer-Mellor OOA are "object oriented." The resulting models can be translated into a variety of implementations, both object-oriented designs and other design styles.

A primary design goal for the Shlaer-Mellor notation was orthogonality, the property that there is only one way to say something, and that having said that something, you have said exactly one thing and nothing else is implied. The consequence is that one fact (about the domain) is stated in exactly one place.

The UML, on the other hand, is a notation for object-oriented designs where the design goal was a richness of expression capable of encompassing all (object-oriented) development phases. The notation is rich enough to capture the Shlaer-Mellor semantics, so it is possible to represent Shlaer-Mellor models using a subset of UML.

The UML does not define a process, though there are plans to add one. Shlaer-Mellor defines a translation-based process. As a consequence, a project can use the Shlaer-Mellor process and gain the advantages of implementation by translation, while at the same time being UML compliant.

Appendix A: Shlaer-Mellor to UML Reference List

The major diagrammatic work products of a Shlaer-Mellor Object-Oriented Analysis model can be represented unambiguously using UML notation. The following table shows the UML diagrams used to represent various Shlaer-Mellor models.

Shlaer-Mellor Work Product	UML diagram
Domain Chart	Package Diagram
Object Information Model	Class Diagram
State Transition Diagram	State Diagram
Object Communication Model	Collaboration Diagram
Thread of Control Chart	Sequence Diagram

Appendix B: Domain-Specific Data Typing in Shlaer-Mellor OOA

Domain-specific data types are defined by providing a name, an appropriate base data type, and certain particular characteristics that are required by the selected base data type, as described below.

An *enumerated data type* is a data type that permits a finite set of values. Define the values, and (optionally) a default.

Use a *symbolic data type* for data elements that have the nature of names. Specify the minimum and maximum number of characters and (optionally) a default.

If a data type is numeric in nature, use a *numeric* base data type. Specify a range, with a low and high limit, and the precision. Specify the units, such as centimeters or degrees Celsius and a (optional) default.

In contrast to a numeric data type, an *ordinal data type* can be used only to express a complete or partial ordering.

Use a *time* or *duration* base type to specify calendar-clock time. Specify a range and a precision.

To define a data type for data elements that represent arbitrary identifiers, use the base type *arbitrary*.

Bibliography

H1: David Harel, *State Charts: A Visual Formalism for Complex Systems,* Science of Computer Programming Vol 8 1987.

M1: E. F. Moore, *Gedanken-Experiments on Sequential Machines*, Automata Studies, Princeton University Press, Princeton, New Jersey, 1956.

SM1: Sally Shlaer and Stephen J. Mellor, *The Shlaer-Mellor Method*, Technical Paper, Project Technology, Inc., 1996, http://www.projtech.com/smmethod/smmethod.html

SM2: Sally Shlaer and Stephen J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data,* Prentice Hall, 1988.

SM3: Sally Shlaer and Stephen J. Mellor, *Object Lifecycles: Modeling the World in States,* Prentice Hall, 1992.