4. **Other internal tasks.** For each internal candidate task activated by an internal event, identify whether any adjacent candidate tasks on the concurrent collaboration diagram may be grouped into the same task according to the temporal, sequential, or mutually exclusive clustering criteria.

The guidelines for mapping analysis model objects to design model tasks are summarized in Table 14.1. In cases where the clustering criterion applies, this means that the analysis model object is designed as a passive object nested inside a clustered task, as described in more detail in Chapter 16.

Because a task might fall into more than one task structuring category, a task will be identified by the first criterion applied successfully to it. Subsequent criteria that apply to the task should either confirm the initial structuring decision or indicate that the decision should be revisited. For example, consider a device interface object that is initially structured as a passive output device interface task but is also activated by a control object. If it transpires that the output operation is
Figure 14.17  Task architecture: Example of initial concurrent collaboration diagram for ATM Client subsystem
14.9.1 Loosely Coupled (Asynchronous) Message Communication

With loosely coupled message communication, also referred to as asynchronous message communication, the producer sends a message to the consumer and continues without waiting for a response. Because the producer and consumer tasks proceed at different speeds, a first-in-first-out (FIFO) message queue can build up between producer and consumer. If no message is available when the consumer requests one, the consumer is suspended.

Consider the initial concurrent collaboration diagram (Figure 14.18a), which depicts the Cruise Control Lever Interface task sending a message to the Cruise Control task. It is desirable to map this message interface to loosely coupled (asynchronous) message communication, as depicted in the revised concurrent collaboration diagram (Figure 14.18b). The Cruise Control Lever Interface task sends the message and does not wait for it to be accepted by the Cruise Control task. This allows the Cruise Control Lever Interface task to quickly service any new external input that might arrive. Loosely coupled message communication also provides the greatest flexibility for the Cruise Control task, because it can wait on a queue of messages that arrive from multiple sources. It then accepts the first message that arrives, whatever the source.

An example of loosely coupled message communication in the Elevator Control System is given in Figure 14.4, which shows the Floor Lamp Interface
resource monitor task. This task receives requests from multiple elevators to switch off floor lamps. The requests are queued FIFO in a message queue. The \texttt{Floor Lamp Interface} task processes the requests in the order in which they arrive.

\subsection*{14.9.2 Tightly Coupled (Synchronous) Message Communication with Reply}

In the case of \textit{tightly coupled message communication with reply}, also referred to as \textit{synchronous message communication with reply}, the producer sends a message to the consumer and then waits for a reply. When the message arrives, the consumer accepts the message, processes it, generates a reply, and sends the reply. The producer and consumer then both continue. The consumer is suspended if no message is available.

Tightly coupled message communication with reply can involve a single producer sending a message to a consumer and then waiting for a reply, in which case no message queue develops between the producer and the consumer. However, it can also involve multiple clients interacting with a single server, as illustrated by the following example.

An example of tightly coupled message communication with reply, involving multiple clients interacting with a single server, is given in Figure 14.19. The clients are \texttt{ATM Client} tasks, which send messages to the \texttt{Bank Server} task, as depicted on the initial concurrent collaboration diagram (Figure 14.19a). Each client needs to
be tightly coupled with the server, because it sends a message and then waits for a response. After receiving the message, the server processes the message, prepares a reply, and sends the reply to the client. The notation for tightly coupled message communication with reply on the concurrent collaboration diagram (Figure 14.19b) shows a synchronous message sent from the client to the server with a simple message, representing the response, sent by the server to the client.

### 14.9.3 Tightly Coupled (Synchronous) Message Communication without Reply

In the case of tightly coupled message communication without reply, also referred to as synchronous message communication without reply, the producer sends a message to the consumer and then waits for acceptance of the message by the consumer. When the message arrives, the consumer accepts the message, thereby releasing the producer. The producer and consumer then both continue. The consumer is suspended if no message is available.

An example of tightly coupled message communication without reply is shown in Figure 14.20. The Sensor Statistics Display Interface is a passive output device interface task. It accepts a message to display from the Sensor Statistics Algorithm.
Control Lever Interface «asynchronous input device interface» task, which then reads the cruise Control Input. This interaction could be depicted as an event signal input from the device, followed by a read by the task. However, it is more concise to depict the interaction as an asynchronous event signal sent by the device, with the input data as a parameter, as depicted on the revised concurrent collaboration diagram (Figure 14.21b).

An example of a timer event is given in Figure 14.22. The digital clock, which is an external timer device, generates a timer event to awaken the Distance
Timer «periodic» task. The Distance Timer task then performs a periodic activity—in this case, calculating the distance traveled by the car. The timer event is generated at fixed intervals of time. The simple message shown on the initial concurrent collaboration diagram (Figure 14.22a) is mapped to a timer event signal on the revised concurrent collaboration diagram (Figure 14.22b).

Internal event synchronization is used when two tasks need to synchronize their operations without communicating data between the tasks. The source task signals the event. The destination task waits for the event and is suspended until the event is signaled. It is not suspended if the event has previously been signaled. The event signal is depicted in UML by an asynchronous message that does not contain any data. An example of this is shown in Figure 14.23, in which the pick-and-place robot task signals the event part Ready. This awakens the drilling robot, which operates on the part and then signals the event part Completed, which the pick-and-place robot is waiting to receive.

14.9.5 Task Interaction via Information Hiding Object

It is also possible for tasks to exchange information by means of a passive information hiding object, as described in Sections 3.7 and 3.8. An example of task access to a passive information hiding object is given in Figure 14.24, in which the Sensor Statistics Algorithm task reads from the Sensor Data Repository entity object, and the Sensor Interface task updates the entity object. On the initial concurrent collaboration diagram, the Sensor Statistics Algo-
The algorithm task sends a simple message, `Read`, to the entity object and receives a `Sensor Data` response, which is also depicted as a simple message (Figure 14.24a). Because the task is reading from a passive information hiding object, this interface corresponds to an operation call. The entity object provides a `read` operation, which is called by the `Sensor Statistics Algorithm` task. The sensor...
Figure 14.25 Task architecture: example of revised concurrent collaboration diagram for ATM client subsystem
Sh1: Shaft Input

Sh1.1: Update

«external input device»
: Shaft

«input device interface»
: ShaftInterface

«entity»
: ShaftRotationCount
Hardware/software boundary

NB: The dashed line for the hardware/software boundary is for illustrative purposes only and does not conform to the UML notation.
NB: The dashed line for the hardware/software boundary is for illustrative purposes only and does not conform to the UML notation.
NB: The dashed line for the hardware/software boundary is for illustrative purposes only and does not conform to the UML notation.
<table>
<thead>
<tr>
<th>«state dependent control»</th>
<th>«data abstraction»</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalibrationControl</td>
<td>CalibrationConstant</td>
</tr>
<tr>
<td>+ processEvent (event)</td>
<td>+ start ()</td>
</tr>
<tr>
<td>+ currentState () : State</td>
<td>+ stop ()</td>
</tr>
</tbody>
</table>
Account

# accountNumber : Integer
# balance : Real = 0

+ open(accountNumber : Integer)
  + credit(amount : Real)
  + debit(amount : Real)
  + readBalance() : Real
  + close()

CheckingAccount

– lastDepositAmount = 0

+ readLastDepositAmount() : Real

SavingsAccount

– cumulativeInterest : Real = 0
– debitCount : Integer = 0
– maxFreeDebits : Integer = 3
– bankCharge : Real = 1.50

+ debit(amount : Real)
+ clearDebitCount()
+ addInterest(interestRate : Real)
+ readCumulativeInterest() : Real

fig 15-10
Reset Trip Speed

Calculate Trip Speed

Reset Trip Fuel Consumption

Calculate Trip Fuel Consumption

Reset Oil Maintenance

Check Oil Maintenance

Reset Air Filter Maintenance

Check Air Filter Maintenance

Reset Major Service Maintenance

Check Major Service Maintenance

«use case package»
Monitoring
UseCasePackage

Driver

Technician

Timer

fig20-03