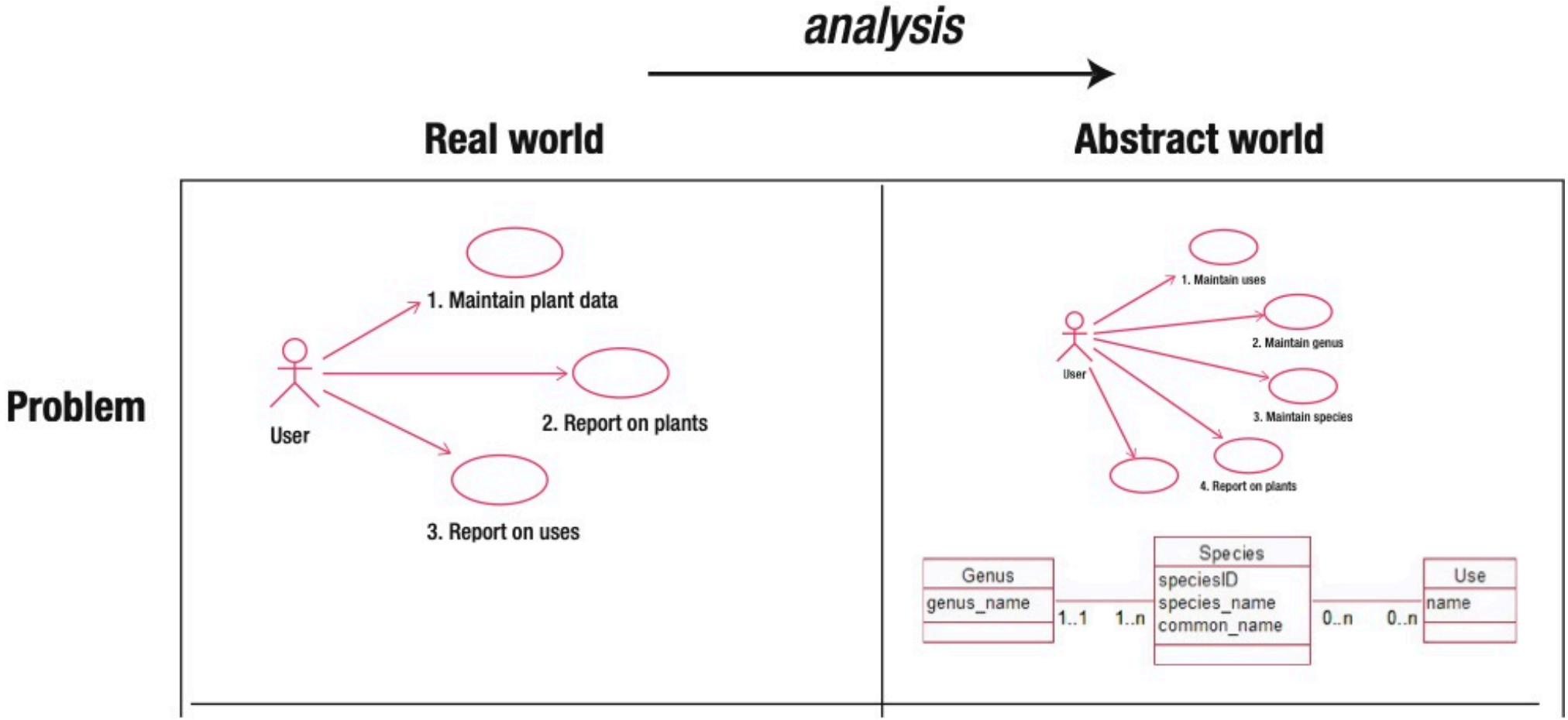
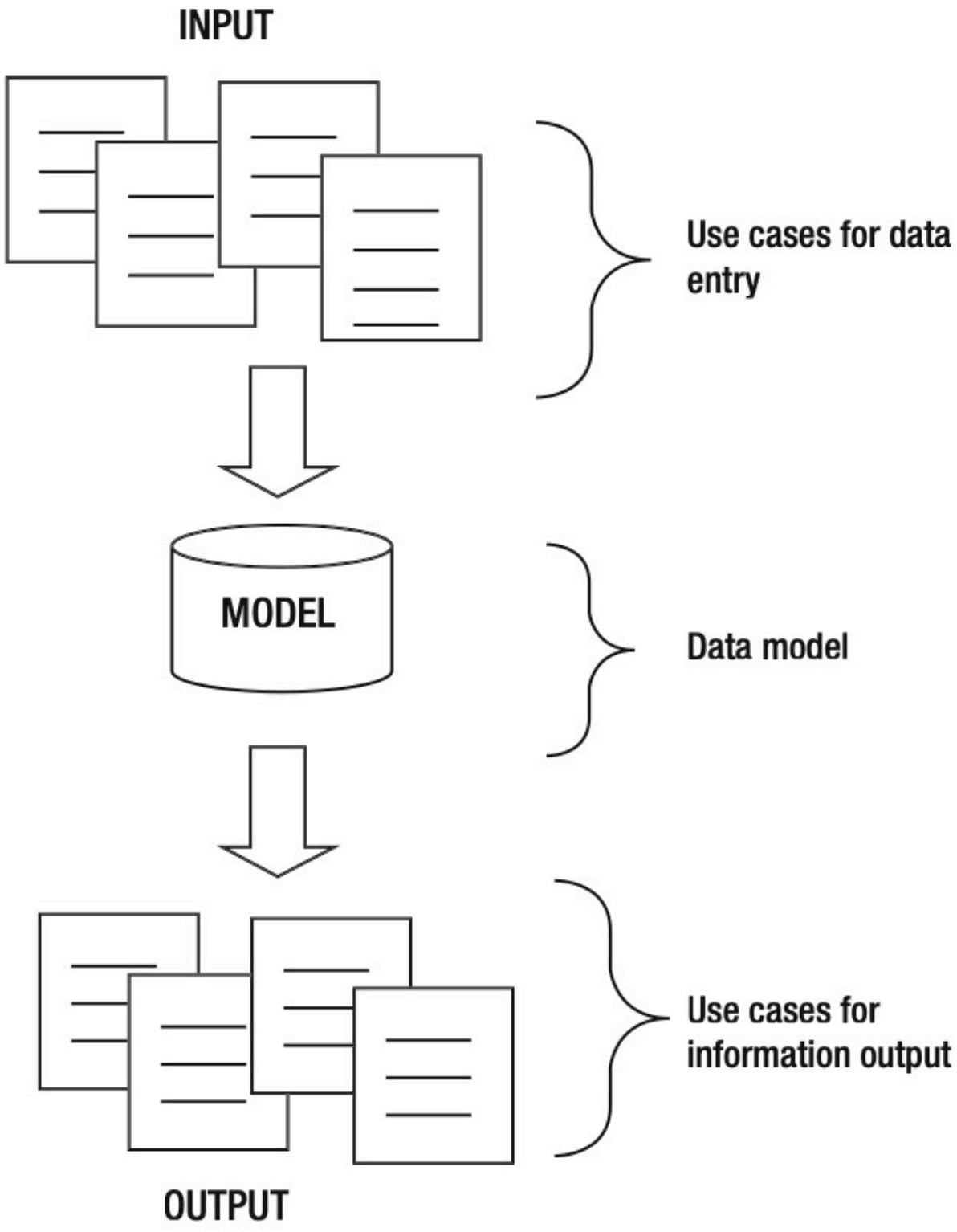
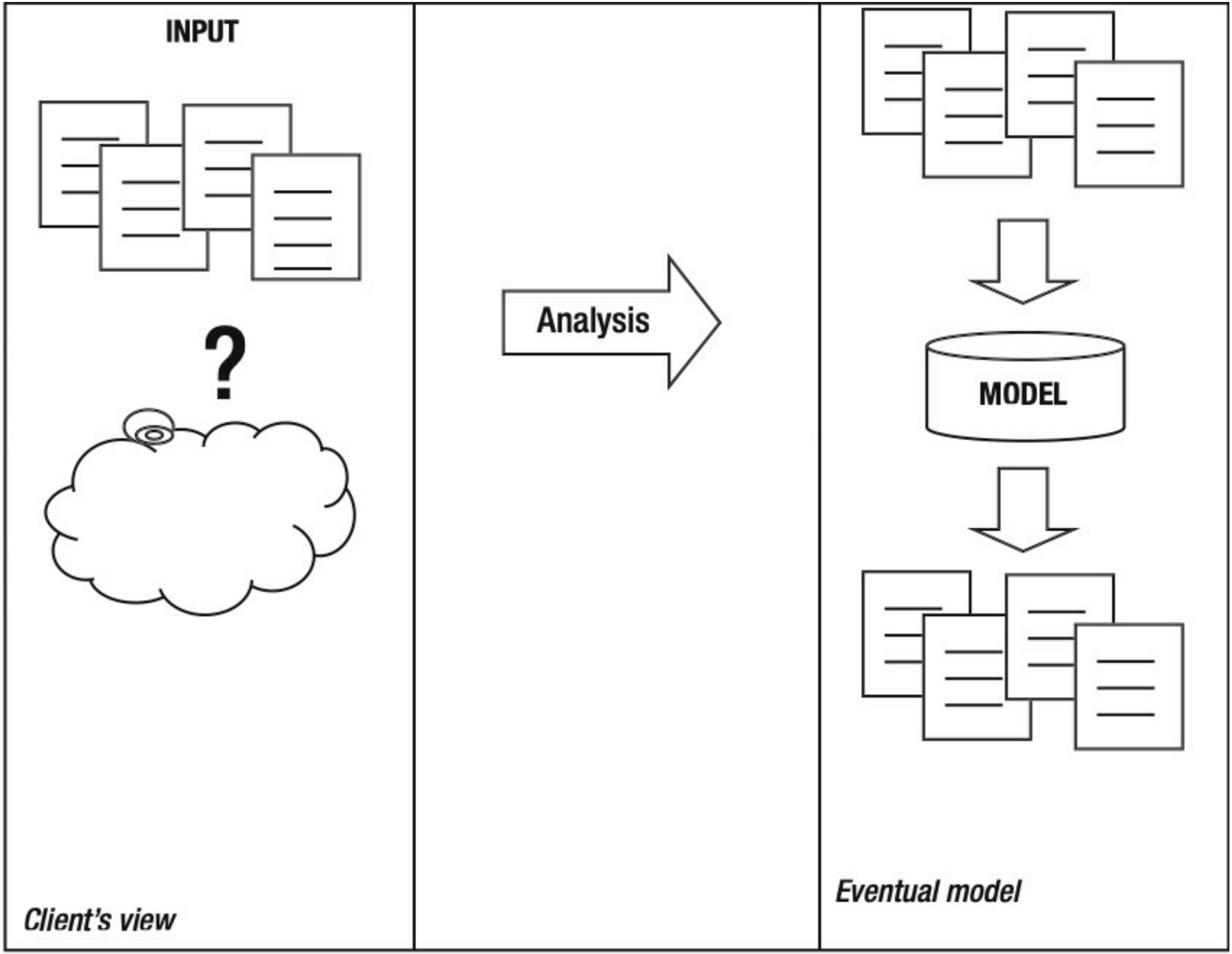


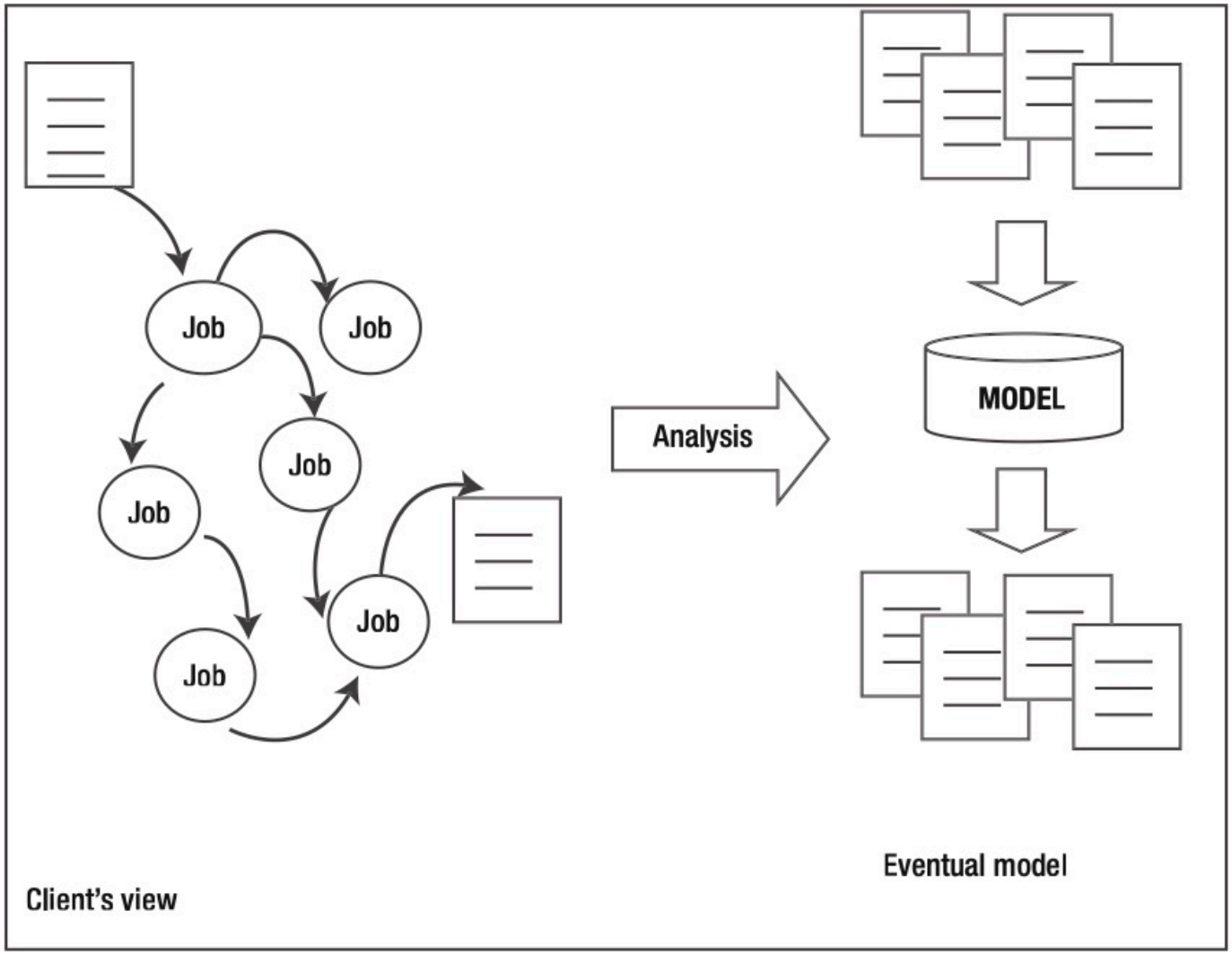
3-1. *The first step: developing an abstract model of the real-world problem*



3-2. *An analyst's view of a typical database system*



3-3. *The analysis of a data-minding problem*



re 3-4. *The analysis of a task automation problem*

A typical description for a task automation problem at a local school might go like this:

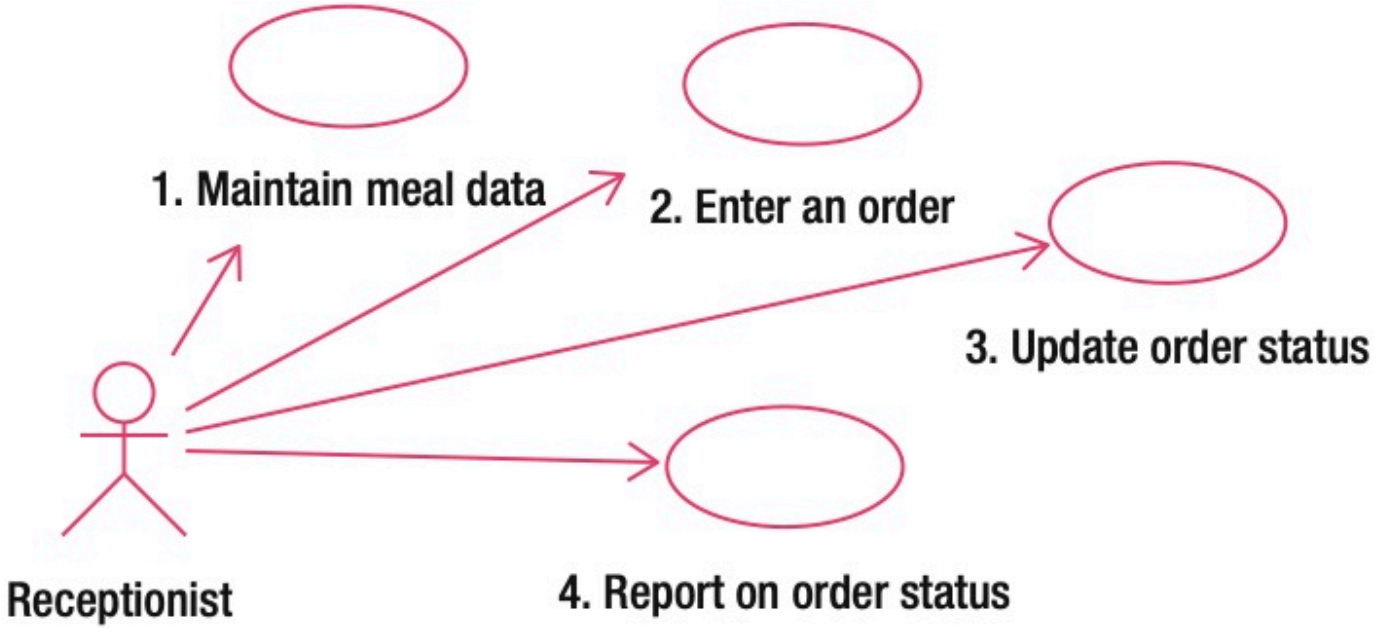
> *When parents call up to say that children are sick, we have to let their classroom teachers know, and if it's sports day and the child is on a school team, the sports teacher might have to sort out substitutes. Then we need to count up all the days missed to put on the child's report. The Department of Education needs the totals each term, too.*

Table 3-1. *Physical User Tasks and Related Data*

Task	Physical Jobs	Data That Could Be Recorded
1	Take order.	Order number, address, phone, name, meals, price, time.
2	Dispatch driver.	Driver's name (or ID?), order number, time, outlets to visit.
3	Pick up meals.	Order number, time of picking up each meal.
4	Deliver meals.	Order number, time of delivery.
5	Enter time sheet.	Anything other than what we already have for each order? Sign–in time, sign–out time?

Table 3-2. *Physical User Tasks for Data Entry and Interaction with the Proposed System*

Task	Physical Job	Interaction with System
0	Record available meals.	Enter and maintain data about each item that can be ordered (ID, description, current price).
1	Take order.	Enter order data (order number, time, address, phone) and the ID of each meal required (assume for now that prices don't change).
2	Dispatch driver.	Record driver's contact number with appropriate order.
3	Pick up meals.	Nothing.
4	Deliver meals.	Record delivery time for the appropriate order (here or possibly at the next step).
5	Enter time sheet.	Nothing.



3-5. *Use cases for meal deliveries*

Use case 1: Maintain meal data. Enter and update data on meals (ID, description, current price).

Use case 2: Enter an order. Enter initial order information (order number, date, address, phone) and for each meal record the ID. (This assumes prices do not change. We will consider price changes later in the chapter in the section "Changing Prices.") Each meal must be one that is already in the system.

Use case 3: Update order status. For a particular order already in the system, add driver contact number or delivery time.

Use case 4: Report on order status. Retrieve all orders satisfying required status (e.g., no driver contact number or no delivery time).

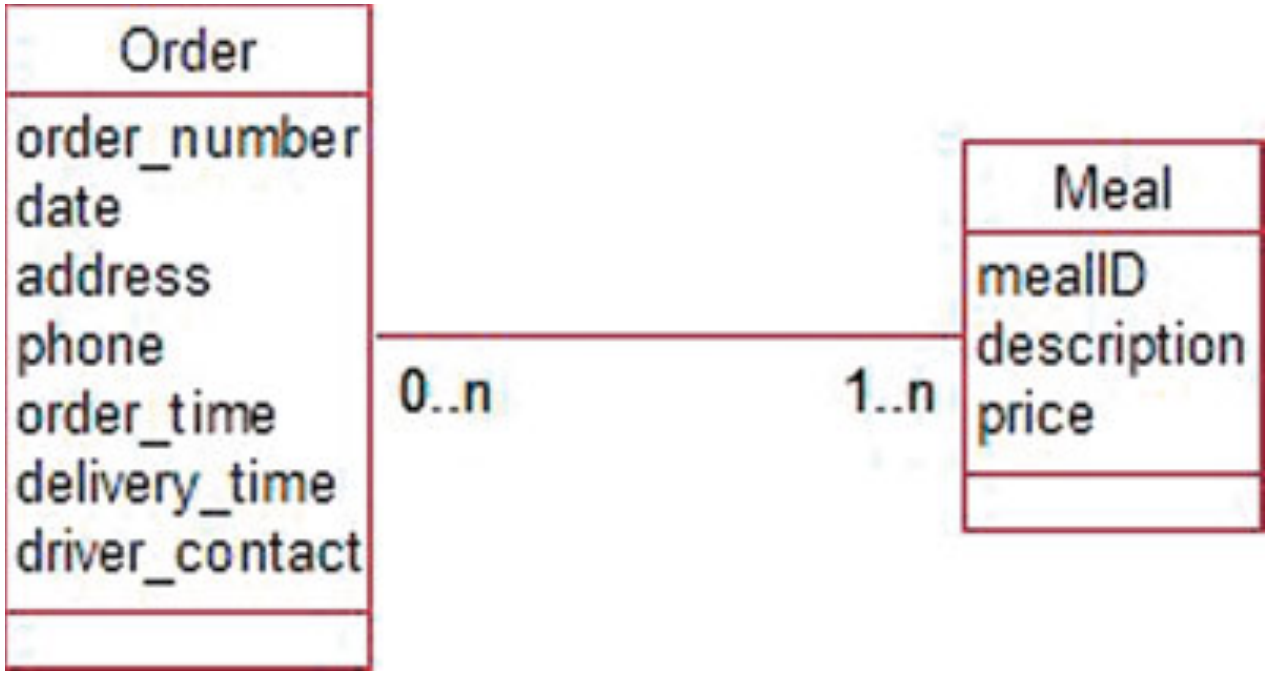


Figure 3-6. *First attempt at a data model for meal delivery database*

In Figure 3-6, we have separated each of the pieces of data we are recording and put them as attributes in the most likely class. Let's recap from Chapter 2 what a model like Figure 3-6 means. Reading from left to right, we have that a particular order (e.g., "to Colombo Street at 8:30 on April 1st") can involve one or more meal types. From right to left, we have that each type of meal (e.g., chicken vindaloo) could appear on many orders but may not appear on any (e.g., no one may ever want to order spinach and anchovy pizza). Just in case there is any confusion, when we talk about a meal, we mean a type of meal as it appears on the menu. We don't mean that a particular portion of curry may end up on more than one order!

ure 3-7. *Use case for reporting statistics*

Use case: Summary reports on orders. (This assumes constant prices.)

For each completed order with a `date` *in the required time period:*

- Find all the associated meals and retrieve their `prices`,
- If required calculate the time of the order by subtracting `order_time` from `delivery_time`.
- If required, group orders by smaller time periods (day, week, etc.).
- Average and/or total prices/times.