CHAPTER

2

Domain Framework for Sales Promotions

Like other business areas, marketing has seen a true information revolution. Effective and efficient use of heterogeneous data has become crucial, especially in highly competitive markets. In this quest for information, the time factor is important: Knowledge is most valuable when it is acquired before the competition gets it. The companies that can design the best marketing campaigns in the shortest period of time are the winners in the marketplace.

Computer systems currently available to companies offer a variety of data retrieval and optimization procedures (such as production planning and supermarket shelf space allocation). Although these systems are often adaptable in parameters, they are mostly static representations of problems that do not fit the dynamic nature of the marketing domain. As new types of data become available in increasing amounts and the human factor in decision making becomes more central, attention seems to be shifting to building adaptable systems that fit the mental models of decision makers.

A product manager is one important marketing decision maker who is responsible for development, pricing, distribution, advertising, and sales promotion of one or more products. Product managers make semistructured decisions, mostly under heavy time pressure. To perform successfully, these managers need well-designed software applications, based on their mental models, to support their decision making.

Mental models of product managers consist of building blocks that are typical marketing concepts—brands, prices, advertisements, sales, market share—or typical marketing actors—customers, competitors, producers, retailers. In this mental counterpart of the real world, the product manager replays, simulates, and forecasts things that happen in order to be able to respond to the opportunities and threats that occur.

In order to facilitate the development of software applications for product managers, we developed a domain framework of the mental models of product managers. The framework focuses on sales promotion decision making, which is a prominent task of product managers. Sales promotions are marketing events aimed at having a direct impact on the buying behavior of consumers. Examples of sales promotions are price-offs, premiums, cash refunds, samplings, and coupons. As the effects of mass media are diminishing, sales promotions are becoming an increasingly important tool for product managers to increase sales [Blattberg-Neslin 1990].

Once constructed, the general domain framework can be reused to build a variety of specific applications for the domain. In a multinational corporation a sales promotion domain framework can serve as the basis for the development of applications for sales promotion decision making in different countries, for different divisions, adapted to the needs of individual managers, and for the various tasks that have to be performed within the domain. The development of such a framework is possible and beneficial because despite circumstantial differences (for example, individual differences in the product they manage, their decision-making style), product managers tend to have similar tasks and share a comparable mental model.

Early reports on object-oriented (OO) frameworks have mainly described system-level frameworks (such as file-access or networking aids) and user interface application frameworks. Recently, research into domain frameworks has emerged, primarily focusing on more-structured and well-studied domains such as accounting or logistics. The framework developed in the present study concerns a semistructured domain. In this domain, as Table 2.1 shows, creativity and experience as well as hard data are important.

Table 2.1 Importance Ratings* of Factors of Sales Promotion Decision Making

FACTORS	AVERAGE
Creativity	6.0
Experience	5.8
Hard data	5.6
Judgment	5.3
Soft data	5.0
Discussions	4.9
Intuition	4.9
Heuristics	4.7
Experiments	4.0
Models	3.0
Theory	3.0

*Ratings obtained from 20 respondents in interviews that we report on later in this chapter (scale: 1–7).

It is the objective of this chapter to describe and illustrate a method to develop frameworks for semistructured domains such as sales promotions. The method has three features that we argue are essential in building such frameworks successfully:

The development method should start with an analysis of the thinking processes of the domain expert—the mental models underlying his or her actions. In this chapter we show the use of think-aloud protocols, which subsequently are analyzed by means of content analysis.

The development method should employ an OO analysis and design methodology in combination with the use of patterns. OO modeling enhances the correspondence between the system and the problem domain. Encapsulation hides the internal details of the framework and inheritance facilitates extension and reuse of the framework. The advantages of OO modeling are further enhanced by using analysis patterns, which are documented solutions to common modeling problems.

The framework should be modeled in a computer-aided software engineering (CASE) environment capable of translating the model into different programming languages. In this way, the framework can be used in many different projects, regardless of specific platform or language needs. Furthermore, using, adapting, and extending the model can be done at the design level (in the CASE environment), preventing important semantic modeling information from being lost [Fayad-Laitinen 1998].

The following section describes this development method in more detail and illustrates it by discussing the development of the sales promotion framework. In *Section 2.2*, a possible application (re)using the framework is shown. In *Section 2.3*, we evaluate our approach, and in *Section 2.4*, we suggest possible extensions.

2.1 Framework Development

Figure 2.1 depicts the development method of the framework and the specific application. Steps 1 to 3, which concern a general approach to the building of the framework, are explained in the next section. In later sections, steps are illustrated for the sales promotion domain. *Section 2.2* shows the use of the framework for the development of a specific client application.

2.1.1 The Framework Development Method Step by Step

The framework development method consists of three major steps:

- Solving sample problems
- Analyzing protocol contents
- Creating the object-oriented framework using CASE tool

These steps are discussed in detail in the following sections.

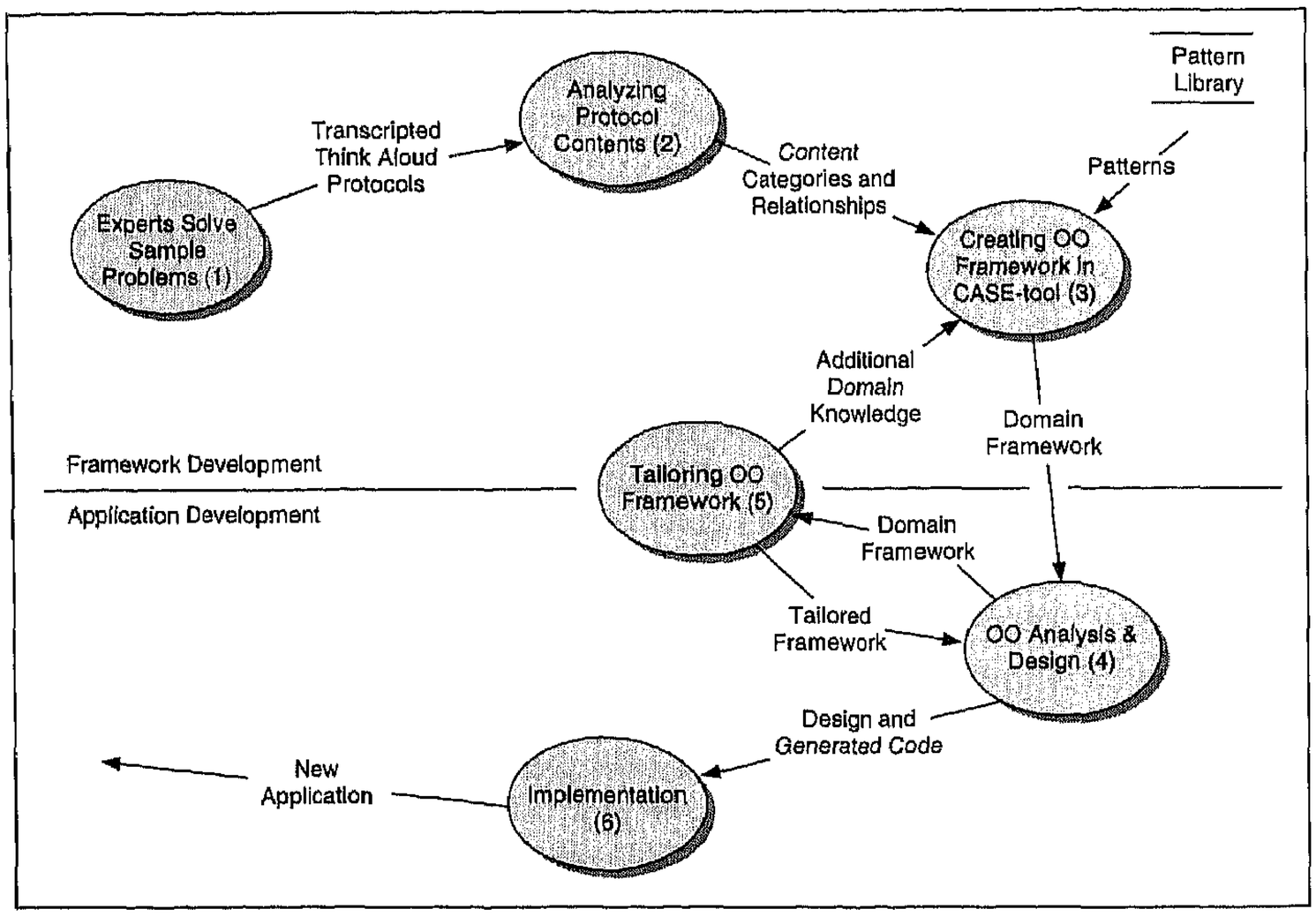

Figure 2.1 Development method.

Step 1: Experts Solve Sample Problems

Developing software to support decision making in semistructured domains without involving domain experts easily leads to incomplete, poorly constructed, and unused software. It has been widely recognized that it is essential to have domain experts involved in OO modeling of software [Rumbaugh 1991; Yourdon 1994]. Framework development also requires input from domain experts, but differs from application development in that end users do not directly benefit from the results. Hence, they are not likely to be available for extensive training and modeling sessions. Also, it is generally agreed that conceptual modeling of a framework is harder and more abstract than modeling an actual application and therefore requires good modeling skills. Conceptual modeling is probably the most difficult and expensive activity performed in the process of framework development.

We propose an indirect yet thorough way to utilize domain expertise for framework development. In this approach, think-aloud protocols of domain experts are used to construct a domain framework. A set of sample problems that can, more or less, be considered representative for the domain are collected or constructed and presented to a group of domain experts. The solutions the experts produce are recorded on tape and transcribed so they can be used for further analysis. The output of this first step is the transcribed think-aloud protocols.

Step 2: Analyzing Protocol Contents

The methodological framework of content analysis is used for the analysis of the protocol data. Content analysis is an accepted research technique in the social sciences and

is concerned with the analysis of the content of communications. A central idea in content analysis is that the many words of a text can be classified into fewer content categories that are presumed to have similar meaning. On the basis of this categorization, the researcher tries to draw valid inferences from the text. For an introduction to this field see [Krippendorff 1980; Weber 1985].

The purpose of the analysis is to find out how domain experts conceptualize (make mental representations of) their domain. In order to achieve this, the protocols generated previously are coded by applying two coding procedures. The first procedure is meant to *discover content categories*. Coders are given instructions to look for nouns related to the domain. The coding units (in other words, the pieces into which the text is subdivided) in this procedure are *word* (such as, "budget") or *word sense* (for example, "the money that is available for the promotion") [Weber 1985], the latter being the part of a sentence that refers to a single entity but is expressed with more than one word.

The coders then classify the units in categories according to equal semantic meaning. For example, "our budget," "the available $300,000," "the money we can spend," can be attributed the same semantic meaning (budget) and put into one category. Each category receives a name that adequately covers the semantics of the units it contains.

We can now rank the categories based on the frequency with which they occur. If several different cases were solved by various experts, the ranked (cumulative) frequency gives us clear insight into the relative importance of the concepts as used by the experts.

The second procedure is performed to find relationships among the content categories. Themes are used as coding units [Weber 1985]. A theme is a unit of text that has no more than the following elements: (1) the perceiver, (2) the agent of action, (3) the action, and (4) the target of action. For example, "I want consumers to pay attention to my product" is divided as follows:

> "I [the perceiver] (want) consumers [agent of action] to pay attention to [action] my product [target of action]."

A relationship consists of two or more themes ("If I want consumers to pay attention to my product *Theme 1*, I could promote by giving an introduction discount *Theme 2*"). All relationships should be coded from all collected transcripts.

Step 3: Creating the Object-Oriented Framework Using a CASE Tool

The Unified Modeling Language (UML) [Booch-Rumbaugh 1997] can be used to model the framework. The OO paradigm holds the promise of being highly suitable for constructing applications for semistructured domains such as marketing, since such domains cannot be modeled in a single proper way and are, moreover, inherently dynamic. These factors impose strong demands on the adaptability of software. The promise of OO to deliver adaptable software systems matches these demands [Gibson-Senn 1989; Korson-McGregor 1990]. Reuse of OO components should enable quick delivery of new system functionality, though adaptability must be explicitly engineered into the system. Frameworks and patterns provide a vehicle to achieve adaptability and reuse [Fayad-Cline 1996]. A framework can be used to build applications by creating new subclasses [Johnson-Foote 1988]. Software developers often lack the

domain expertise that is encapsulated in the framework. The knowledge incorporated in a domain framework exceeds that of other reusable components. Unlike other reusable components, OO frameworks incorporate object interaction and define default behavior [Taligent 1997].

Most OO analysis methods suggest that analysts should mark all the nouns they find in domain-related texts as candidate classes. In semistructured domains, this easily leads to unmanageable numbers of classes.

In our method, this problem is tackled by the initial reduction of the set of nouns by putting them into content categories (Step 2, first procedure), and by a further reduction of the initial number of classes by mapping only the top 20 or 30 content categories to object classes. This mapping is the first step in the actual construction of the framework. The exact number of categories included in the initial object model can be decided by the developer while taking into consideration factors like the estimated size of the framework and the distribution of the frequencies over the categories. The classes selected in this way make up the initial object model, which, thus constructed, forms the basis for the rest of the modeling process.

The modeling process proceeds by refining this initial framework using the relationships that were found in the protocols (Step 2, second procedure). Within relationships, verbs relate one word or word sense to another, implying the way classes are related to one another. The text of a relationship is mapped to the OO model by (1) checking whether the classes the relationship refers to are correctly represented in the initial object model and (2) modeling relevant verbs either as relationships between objects or as behavior.

It is hard to delimit the inferences that can and cannot be made by the modeler. There are two important criteria that can be set to judge modeling choices [Krippendorff 1980]. First, the modeler should aim at as little impairment of the original text as possible. Second, inferences based on few relationships yield a lower degree of certainty than those often encountered, indicating more stable relationships. The modeler should administer the modeling choices that are made, for instance, by storing them in tables or by administering them directly in the CASE tool, thus relating source text references to classes, attributes, relationships, or behavior.

2.1.2 Using the Development Method for the Sales Promotion Domain

This section shows how to apply the framework development method in the sales promotion domain.

Step 1: Experts Solve Sample Problems

We devised two sample problems representative of the domain of sales promotions. Each interviewee (10 product managers and 10 sales promotion authorities) had to solve a problem on the design of a sales promotion plan for the introduction of a new salad dressing (see Exhibit 2.1) and another one in which a sales promotion plan had to be devised for a troubled fashion magazine (not displayed here). The respondents were asked to think aloud so their solutions could be recorded and transcribed.

EXHIBIT 2.1
PRESTO-DRESS SAMPLE PROBLEM

Presto-Dress is a new salad dressing. The producer has developed a new package concept using a transparent plastic box with a handy spout that allows the user to dose the salad dressing very accurately. The market for salad dressing has two submarkets: the regular dressings market and the special dressings market. Goals are "making Presto-Dress a flourishing brand in the special dressings market, in terms of market share and profit,"' and "to promote brand-switching from Creamy-Dress (a product in the special dressings market that the producer wants to eliminate) to Presto-Dress."

The assignment: Design a sales promotion plan for the introduction of Presto-Dress. The budget is $300,000.

Step 2: Analyzing Protocols Contents

The purpose of the interviews and the analysis of the protocols was to find out how sales promotion managers conceptualize (make mental representations of) their domain.

Exhibit 2.2 displays a small piece of protocol transcript collected in Step 1. The 20 transcripts were coded by business school students enrolled in a course on sales promotion decision making. Each interview was coded by two students to assess coding reliability.

The first coding procedure was meant to discover the important concepts in the sales promotion domain. The instructions were to look for nouns, either related to products or to sales promotions, in order to simplify the coding process and reduce the number of categories beforehand, eliminating irrelevancies. Table 2.2 shows the top 30 content categories, indicating the number of units in each category. Brand, product, and promotion name appeared so frequently that counting them would have distracted the coders (labeled N/A in Table 2.2).

Categories with a high frequency for one problem and a zero frequency for the other mark problem-specific concepts. For example, the concept of Package is relevant only for

EXHIBIT 2.2
PART OF A DOMAIN EXPERT PROTOCOL

Well, I think, you could . . ., I would wonder whether I want a promotion the first year at any rate? I mean, since it is a new product. It is new. The package is great. I'm going to campaign, I could imagine to . . ., well, because what I really want is to drive consumers to make trial purchases. Well, if I'm going to campaign, I'd probably promote by giving an introduction discount; so no added value promotions. That would distract attention from the product since added value is already in the product. I mean, suppose I would give a CD that would, . . ., no . . ., I want to focus on my product and drive people to buy the product. If I want consumers to pay attention to my product, I could promote by giving an introduction discount. Or I would do a refund promotion. In other words: Give the consumer cash money back immediately, or . . ., I could promote by refunding the entire product price if people send in the bar code of the product.

Table 2.2 *Ranked Average Number of Occurrences of Categories in Protocols (Number of Protocols = 20)*

RANK	CATEGORY	PRESTO-DRESS	MARIE-FRANCE	AVERAGE	RANK	CATEGORY	PRESTO-DRESS	MARIE-FRANCE	AVERAGE
1	Brand	N/A	N/A	N/A	16	Market	24	12	18
2	Product	N/A	N/A	N/A	17	Shape of Package	27	0	14
3	Promotion Name	N/A	N/A	N/A	18	Promotion Strategy	27	0	14
4	Promotion Budget	68	86	77	19	Character	0	17	9
5	Sales	3	102	53	20	Proposition	16	0	8
6	Promotion Costs	40	52	46	21	Producer	11	2	7
7	Promotion Goal	61	29	45	22	Distribution	11	0	6
8	Consumer	19	56	38	23	Added Value	10	0	5
9	Advertiser	0	71	36	24	Reach	5	5	5
10	Price	30	29	30	25	Promotion Communication	10	0	5
11	Package	58	0	29	26	Awareness	7	2	5
12	Trial	44	4	24	27	Promotion Prospect	0	9	5
13	Ad	24	20	22	28	Product Content	4	4	4
14	Market Share	35	3	19	29	Profit	8	0	4
15	Target Market	3	34	19	30	Promotion Location	0	8	4

the salad dressing problem, and the concept of Advertiser is relevant only for the troubled fashion magazine (see Table 2.3). These categories are deliberately included in further analysis, since they indicate important problem-related concepts. When we design the framework, such concepts point at *hinges* of the framework (in other words, places where adaptability has to be engineered into the framework explicitly) [Fayad-Cline 1996].

With the second coding procedure we intend to *find relationships among the content categories*. Table 2.3 shows relationships and their references to content categories that were extracted from the protocol fragment displayed in Exhibit 2.2.

Step 3: Creating the Object-Oriented Framework in a CASE Tool

Marketing decision making is strongly oriented toward certain common concepts and actors with characteristics and behavior. A strong mapping between the real world and potential software objects exists. As is noted in the marketing literature [Wierenga-Van Bruggen 1997], managers are not willing to accept systems based on models that are at

Table 2.3 Sample Relationships

REFERENCE	RELATIONSHIPS	CONTENT CATEGORIES
ns1-1	Do I want a promotion the first year, since I'm dealing with a new product here?	Promotion, Date, Product
ns1-2	I'm going to do a promotion, since I want consumers to make trial purchases.	Promotion, Consumer, Trial Purchase
ns1-3	If I'm going to campaign, I'd probably promote by giving an introduction-discount.	Promotion, Product Introduction, Introduction Discount
ns1-4	No added-value promotions because the added value is already in the product.	Added-Value Promotion, Product Added Value, Product
ns1-5	If I want consumers to pay attention to my product I could promote by giving an introduction discount.	Consumer, Attention, Product, Promotion, Introduction Discount
ns1-6	If I want consumers to pay attention to my product I could do a refund promotion.	Consumer, Attention, Product, Refund Promotion
ns1-7	If I want consumers to pay attention to my product I could promote by giving an immediate refund.	Consumer, Attention, Product, Refund
ns1-8	I could promote by refunding the entire product price if people send in the bar-code of the product.	Refund, Product Price, Bar Code

variance with their own mental model. Object orientation makes it possible to build systems that closely match reality as perceived by its users. Also, product managers act in a dynamic environment where products and actors quickly appear, disappear, and change position. The promise of OO to deliver adaptable software systems matches this dynamic nature of the domain.

We started building the sales promotions framework by mapping the top 30 content categories (see Table 2.2) to object classes. This initial object model forms the basis for the rest of the modeling process.

We subsequently refined the framework with the use of the relationship that we found in the protocols of the sales promotion decision makers (Step 2, Table 2.3). Figure 2.3, for example, displays an inheritance relation between Purchase and Trial. This link was created during modeling when we came across the statement, "I'm going to do a promotion since I want consumers to make trial purchases." (see Table 2.3 [ref. ns1-2]). In the initial model (top 30 classes), only the class Trial was modeled. By analyzing the relationships, we found that a trial purchase is a specialized version of a regular purchase; hence, we created Purchase and made Trial a specialization of this class.

The sales promotion domain framework model is too complex to remain comprehensible while being displayed in one diagram. The model is decomposed into six smaller units (see Figure 2.2)—so-called packages—that can contain all kinds of modeling elements [Booch-Rumbaugh 1997]. In partitioning the model, we strove for high cohesion based on grouping classes addressing similar domain aspects and loose coupling between distinct classes. Arrows between the packages indicate visibility relationships: If a client object in the Product package wants to make use of a server object in the Consumer package, then a visibility relationship must exist between the client and server classes [Fowler 1997]. The Consumer and Product packages have mutual visibility. The framework is too extensive to be described here entirely. To illustrate both the generic as well as the domain-specific classes of the framework, we highlight the contents of the Consumer, Promotion, and QuantityHistory package in this section.

Figure 2.3 displays an explosion of the Consumer package. Consumer is one of the central concepts of the framework and captures data on individual consumers and their purchase histories. The interview results indicated that sales promotion managers have a high interest in data on the level of the individual. Sales promotions and direct-marketing activities (telemarketing, direct mailing) have increasingly been linked and merged. Producers can buy individual data available either from scanners in stores (linked to con-

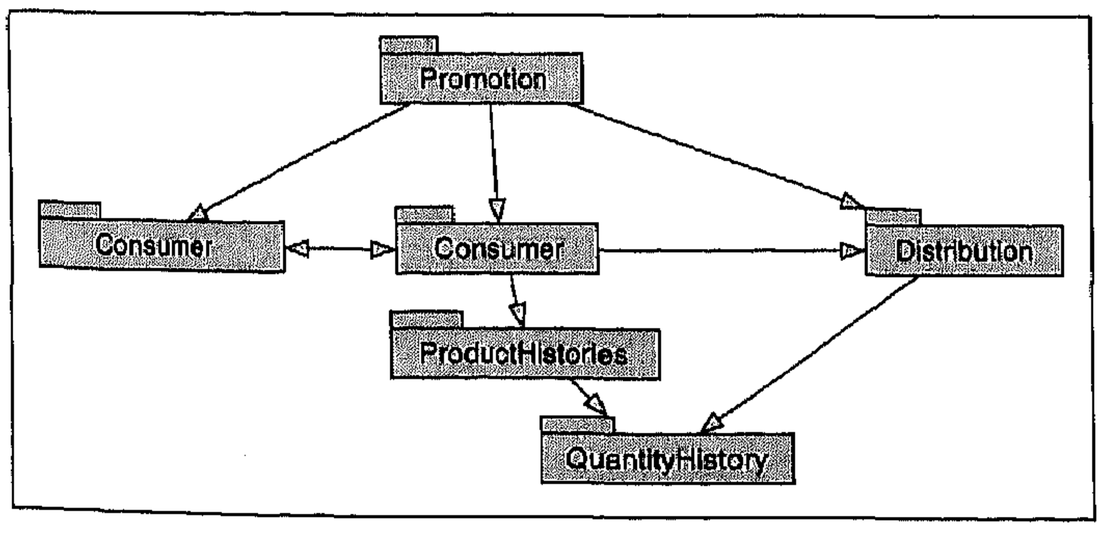

Figure 2.2 The packages and their dependencies.

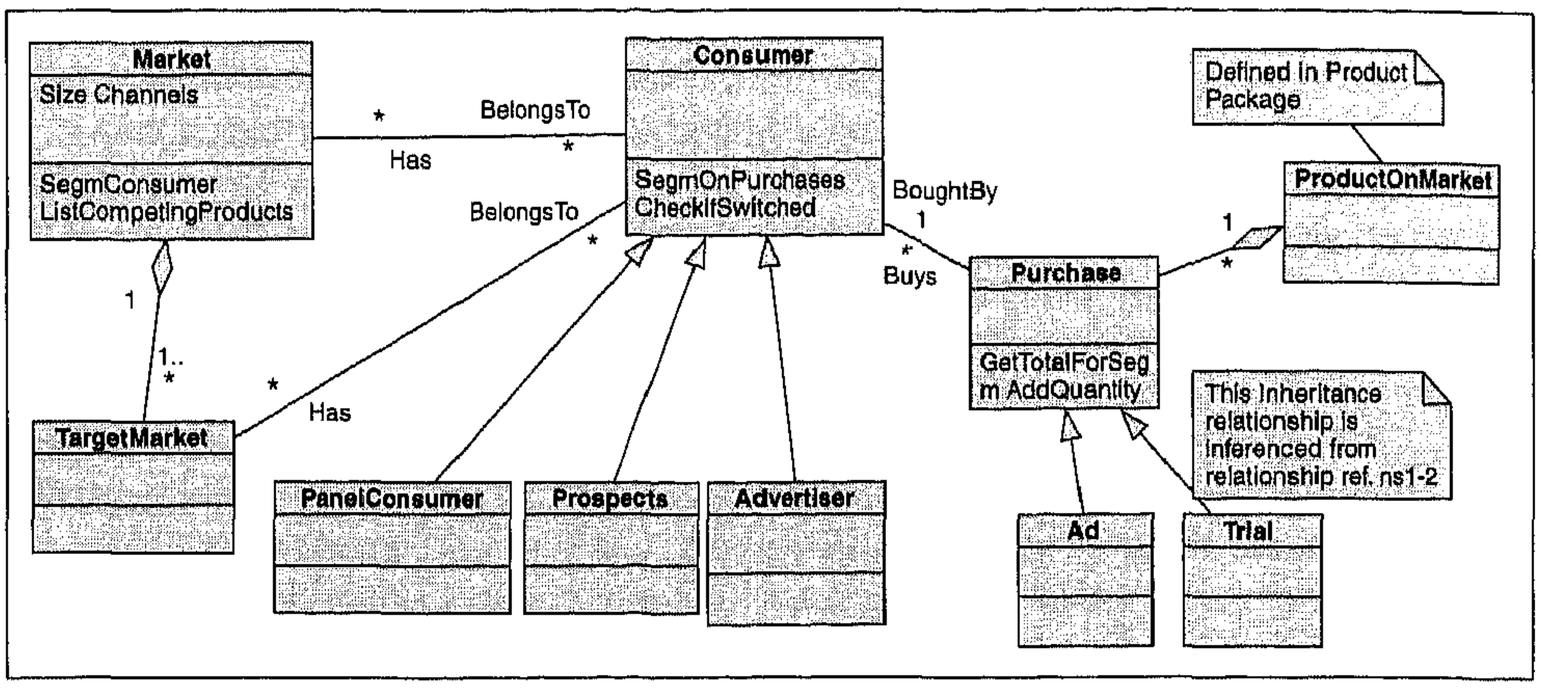

Figure 2.3 The Consumer package. The frequencies of the corresponding categories are displayed in the top-left corner of the class.

sumer data by electronic cards) or from consumer panels (samples of consumers whose purchases, media consumption, preferences, and so on, are recorded over a longer period of time). This individual data can be represented by the Purchase class. The subclasses Ad and Advertiser illustrate how a simple extension to the framework enables the manager to store data on a special consumer: an Advertiser that buys an Ad in a Magazine.

An example of framework behavior that was modeled in the Consumer package pertains to the activity of segmenting. Many relationships extracted from the protocols addressed this prominent marketing strategy, which concerns the finding groups of more or less homogeneous consumers according to some prespecified characteristic(s). Figure 2.4 zooms in on this behavior in the Consumer package, by showing how the manager interacts with the system when he or she wants to segment consumers, based on their Purchases.

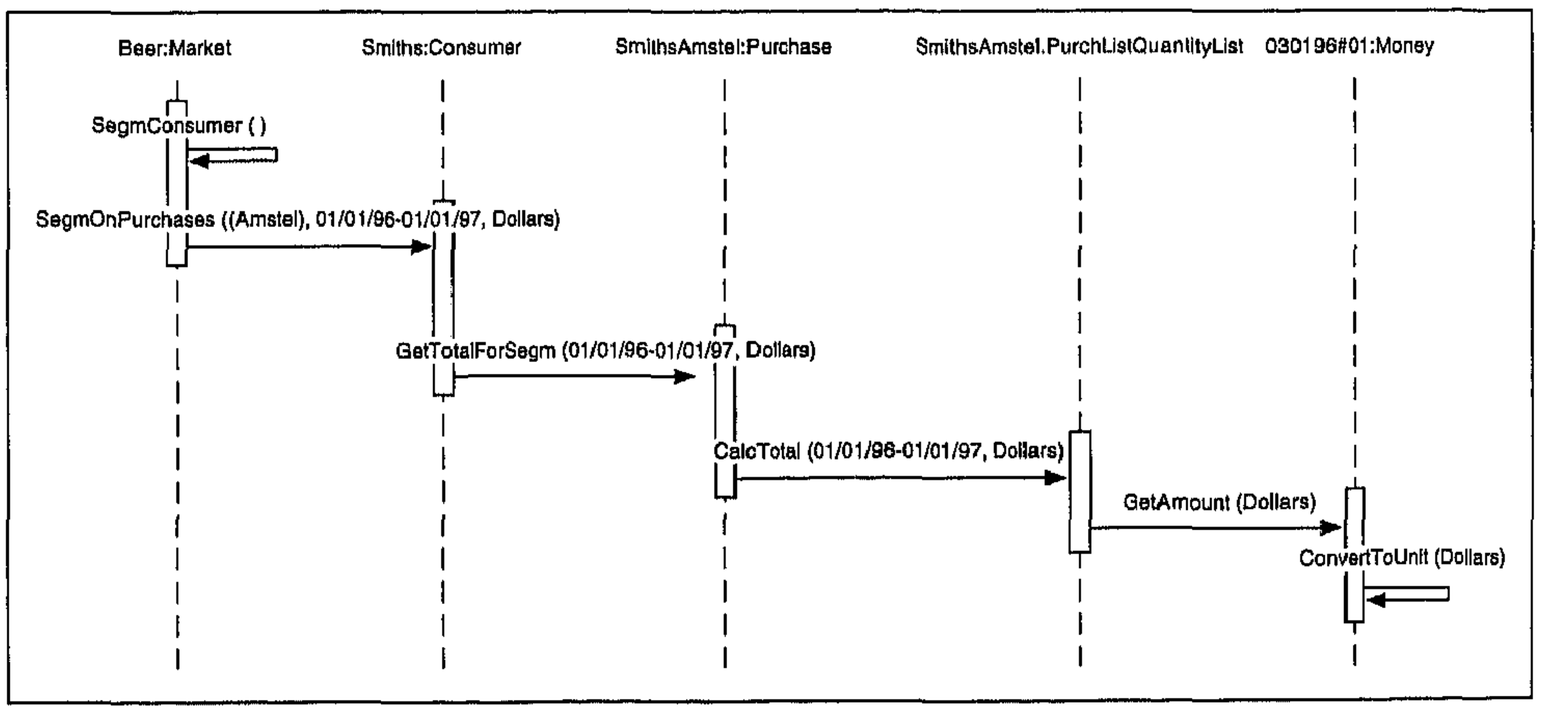

Figure 2.4 A sequence diagram of how the framework handles segmenting consumers. Note: Interaction with files and GUI components are not shown in order to simplify this figure.

Suppose a product manager of Amstel Beer wants to gain insight into how consumers are grouped over categories based on how much Amstel they purchased in the year 1996 (for example, divide them into three segments: heavy, medium, and light users).

First, the manager is presented a list of attributes by which consumers can be segmented (method: SegmConsumer). In this example, the manager chooses the variable Purchases, also indicating the period and currency he or she wants to consider. This request generates a message that is sent to each consumer of Amstel (in the diagram the message is sent to consumer Smiths, invoking the method SegmOnPurchases, passing the arguments 01/01/96–01/01/97, Dollars). Each consumer object now retrieves its total purchases of Amstel in that year (message to Purchase objects, invoking the method GetTotalForSegm, arg 01/01/96–01/01/97, Dollars). If necessary, the currency is converted to Dollars (message to Money objects, invoking the method ConvertToUnit, arg Dollars). The CalcTotal message illustrates how polymorphism is used, since the message can be sent to any consumer characteristic for which a history is kept (for example, TotalExpenditures, NrOfVisitsToStore).

Figure 2.5 shows an explosion of the Promotion package and displays the sales promotion types the managers commented on in the interviews. The promotion types were grouped based on similarity in characteristics and behavior, as extracted from the protocols. Promotions based on price discounts, for example, were put in one group, since they all share a mechanism that influences the product price. The other sales promotion types influence the added value of the product. A (virtual) pricing method (Pricing) was defined on the level of the abstract class (Price) and was overridden at the subclass level: Discount directly lowers the price, Coupon discounts the price when a coupon is handed over, ProductPlus promotions give an extra quantity of the product for the same price, and Refund gives a discount on the sale directly or after send-

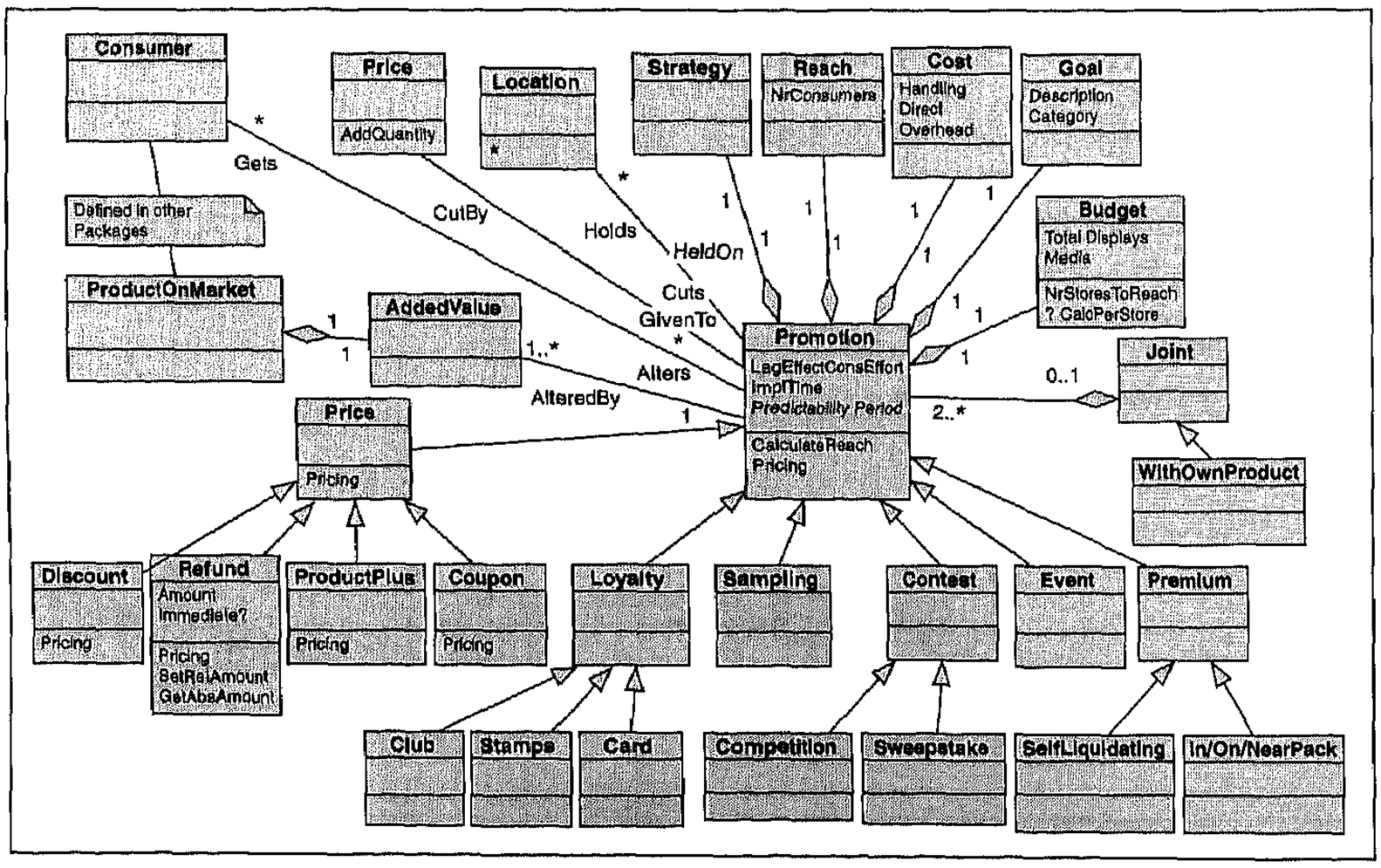

Figure 2.5 The promotion package.

ing in a proof of purchase. The Pricing method is another example of how polymorphism is used in the framework.

A special type of promotion often used by our interviewees was the joint promotion. A joint promotion can be any type of promotion and is a coordinated effort among several different producers. The solutions managers proposed to the Presto-Dress sample problem (see Exhibit 2.1) often had a joint character: giving away samples of the salad dressing when consumers buy lettuce, or gluing coupons on packages of other products of the producer's product range (WithOwnProduct).

The managers choose a promotion type based on the goal of the promotion and, among other things, the available time and the budget. In specifying their promotion they set attributes such as the period in which the promotion will be executed, the distributors with whom they will operate, the trade conditions, the displays they will provide to the stores that adopt the promotion, media expenditures, and many others, not displayed in Figure 2.5 for reasons of clarity.

The Quantity History package shown in Figure 2.6 includes classes that allow the storage of a variety of data over time. The interview data showed that product managers depend heavily on real-world data such as product sales, consumer purchases, market share, and awareness. Variations in product and market require different units of measurement. For example, a bottle of spring water has a price in U.S. dollars and a volume in gallons in the United States, whereas in the Netherlands guilders and liters would be used. These diverse measurements are not present in the programming systems currently available. Anticipating that many analysts have to deal with this problem, Fowler proposed Quantity patterns: analysis patterns describing solutions to this problem based on his experience in modeling hospital systems [Fowler 1997]. As suggested in these patterns, a general Quantity class is introduced, containing a number and a unit of measurement. Specific quantity types we needed for the framework were subclassed. The Money class, for instance, contains Amount and (a currency) Unit. Typical needs of the Money type, such as displaying the quantity using a fixed-point notation, are implemented in the Money class itself. We elicited the subclasses of Quantity from the protocol data: Share (for example, market share), BooleanFactor (for example, a consumer is aware or not aware of a product), and different units for mea-

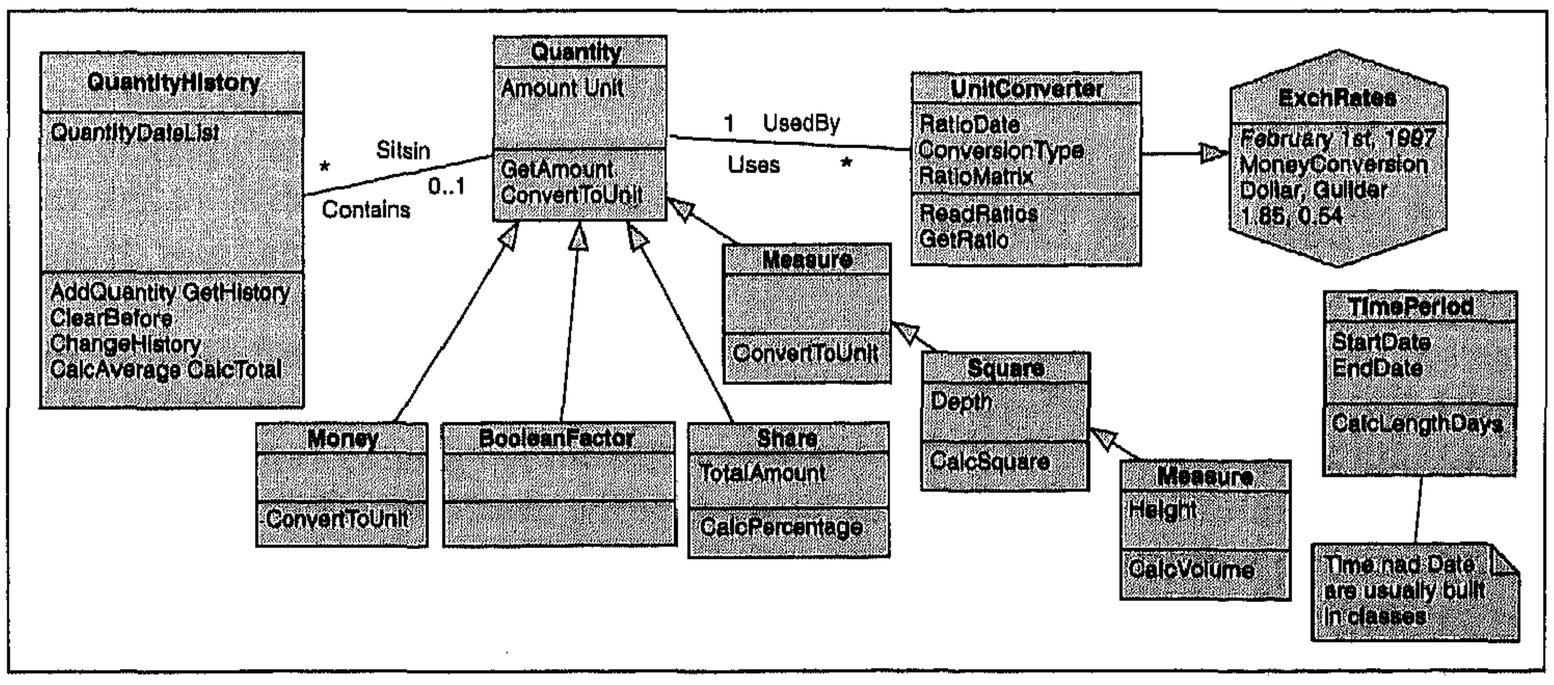

Figure 2.6 Quantity History package.

suring sizes and contents. Product managers look at data from different countries and view quantities on different aggregation levels. This requires conversion of quantities to other units. As suggested by Fowler, a UnitConverter class is designed to perform such conversions. Figure 2.6 shows a sample object, ExchangeRates, that performs such unit conversions for Money objects.

The manager is not only interested in current information, but tends to base many of his or her decisions on historical developments—that is, product sales in the last six months. A system built for the sales promotion domain must be able to store a variety of quantities over a period of time. The QuantityHistory class was designed based on Fowler's Historic Mapping patterns. This class can contain any number of objects from the quantity hierarchy and the dates at which the quantities are valid. The class contains methods like CalcAverage, a method that calculates the average over the Quantities within a specified time period. Many classes in the framework—for example, Purchase and Awareness—are descendants of QuantityHistory and thus are able to store histories data.

The framework was modeled in MetaEdit, a CASE tool that supports UML, which allows methodology engineering and methodology adaptation, and generates source code in different programming languages such as C++, Smalltalk, and Delphi. In case source code cannot be generated for the programming environment the system is built in, custom instructions can be programmed in the CASE tool in order to do so [Smolander 1991]. We discovered that using the CASE tool (in combination with UML) had a number of advantages. First, UML support gave us a combination of well-known diagramming techniques. Useful in handling the complexity of design, for instance, the package construct serves as an entry point to lower-level diagrams. Second, the meta-level adaptability permits extra documentation fields (such as extra comments, references to protocol relationships) to be added to UML concepts. This domain information is crucial to enable future users (developers) to understand the framework. Third, source code generation facilitates adaptation of the framework on the design level whenever a new application is being created. The class operations are implemented in a pseudocode. CASE tools exist that allow source code (or a generic formal language) to be entered directly into the CASE environment. However, including this level of detail into a framework can easily distract the designers from the domain issues and may destroy the advantage of working at a higher level of abstraction. Delivering the framework as programming code would cause a loss of much semantic information.

2.2 Application Development

This section will briefly describe Steps 4 through 6 of the development method (see Figure 2.1) by describing the Sales Promotion Intelligence Data EngineeR (SPIDER) sample application.

The sales promotion framework can be used for the development of a whole range of applications. Software can be designed for different tasks that need to be performed in relation to sales promotions, such as an application for calculating the profitability of sales promotions or time scheduling of different promotional events through the year. The framework can also be used to build sales promotion software for different

industries like banking, automobiles, or fast food. The sample application we developed is SPIDER, a system that can store and retrieve sales promotion events for managers who need to design sales promotion plans for products in fast-moving consumer-good markets. The remainder of this section elaborates on this system.

Product managers can gather a lot of experiential data on their sales promotion activities. Most managers manage multiple brands, each of which has multiple variants (different sizes, flavors, packages). Each product has its own promotional agenda that often comprises multiple promotional events a year, executed in different markets, in different regions, and for different customers. Promotional events are considered great learning opportunities by managers. One problem with the available promotion data is that the information is scattered over various sources throughout the organization: databases, market research reports, videotapes, documentation maps, and the manager's mind. In markets where competition is fierce, decisions have to be made fast and information has to be available instantly. A product manager is generally unable to recollect all relevant experiences that can be used for decisions.

For SPIDER, we needed both a technique to represent the domain and its knowledge and a reasoning mechanism for using that knowledge in the proper way. The sales promotion framework was used for the representation of the domain. This provided the developers, for instance, with the semantics for storing data, organizing it on the basis of the domain knowledge. Promotion Package objects (see Figure 2.5) are used to store data on the promotion itself (for example, Type, Budget, Media), Consumer Package objects (see Figure 2.3) store data on consumers who participate in the promotion (such as Name, Address, AmountPurchased), Distribution objects store data on the retailers that adopt the promotion (for example, Allowance, NrOfStores), and so on.

SPIDER is based on the problem-solving technique known as case-based reasoning. Put simply, case-based reasoning is the computerized equivalent of what we refer to as reasoning on the basis of analogy when it is performed by humans. New problems are solved by using previously designed solutions to similar problems. A case-based reasoning system can serve as a central repository in which experiences can be collected. Algorithms are available for retrieval of the right cases at the right time. Certain case-based reasoning systems have functionality for adapting cases to the new problem situations, which is useful since problems are seldom identical. [Aamodt-Plaza 1994; Schank 1982] provide a good introduction into the field of case-based reasoning.

The domain framework is used by SPIDER, by having its data stored in, and retrieved from, the framework's objects. Objects once defined (as specific consumers, products, and retailers) can remain stored in the system, making entry of new cases an easy job. Most data can be reused: A new sales promotion occurs in an already defined market, was held by a known competitor, and so on.

Figure 2.7 shows the basic architecture of SPIDER and its use of the sales promotion domain framework.

The system is implemented in KAPPA-PC, a hybrid environment combining multiple representation paradigms, including objects, rules, and procedures. The CASE tool we used (MetaEdit) could not generate code for this environment, but provides the possibility to customize code generation. We used this possibility and put in custom-made instructions that are used by the CASE tool to generate KAPPA-PC code.

We performed a UML-use scenario analysis [Booch-Rumbaugh 1997] for finding out the scope and required functionality of SPIDER. The use scenarios comprise the basic

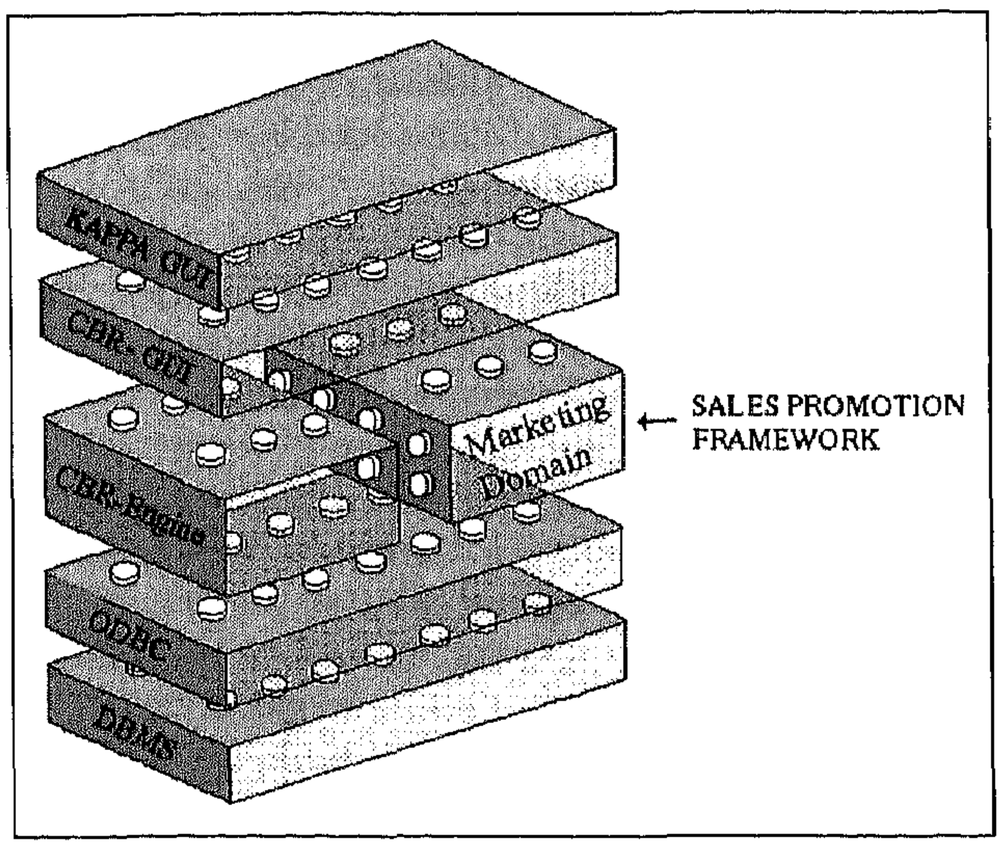

Figure 2.7 Architecture of SPIDER.

functional elements of the system (Exhibit 2.3 shows a textual example of such a scenario). The scenarios were input for the rest of the modeling process: drawing class diagrams and interaction diagrams (see Figure 2.1, Step 4). After this stage, an assessment is made of how the framework can be tailored for the specific application (Step 5). Then, modeling trajectory source code was generated and final implementation issues were addressed (Step 6).

2.3 Lessons Learned

We learned a number lessons in applying the framework development method.

As opposed to our initial expectations, new candidate classes were still found after analyzing a substantial number of expert protocols (see Figure 2.8). We think this will be the case in many semistructured domains, since there is not an objective, fixed set of

EXHIBIT 2.3
AN EXAMPLE USE SCENARIO FOR BUILDING SPIDER

The user selects *Improve brand image* and *Increase loyalty* as values for the *Goals* attribute: The attribute *Goals* is a multivalued text attribute with different options to choose from. As soon as the user clicks the mouse over such an attribute, a list becomes visible, displaying the options from which the user can choose. This attribute allows the selection of multiple items from the list (such as *Improve brand image* and *Increase loyalty*). In case multiple items are selected, the user can express the wish to enter weights for the different items. If the user does so, each item will be displayed along with a slider, ranging from 0–100, with which he or she can set the weights.

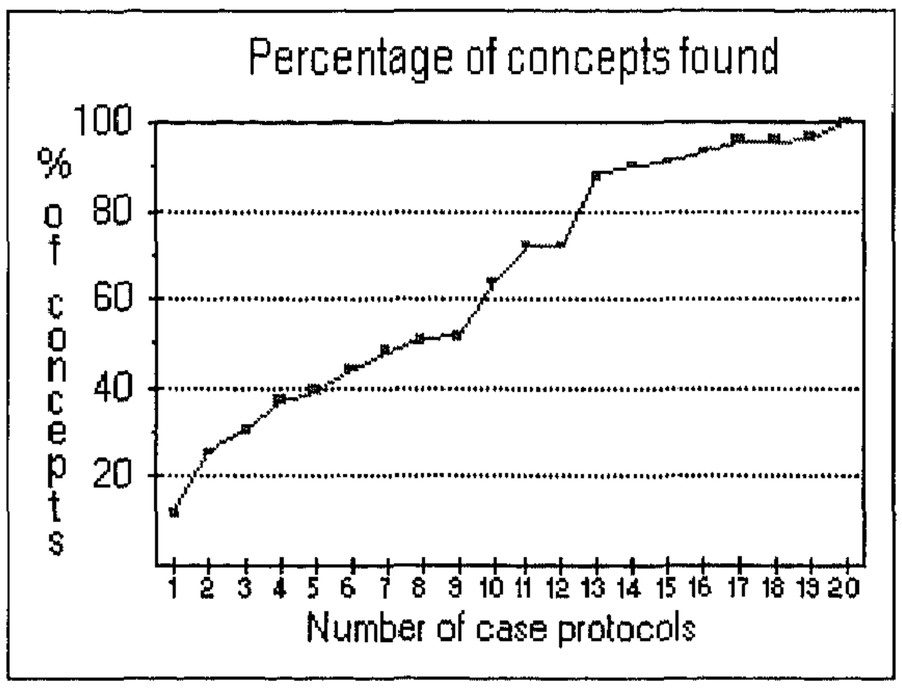

Figure 2.8 The percentage of concepts found against the number of protocols.

elements that play a role. Structured domains, on the other hand, have such a set, making modeling more easy. Ninety-five percent of the candidate objects were found after analyzing 17 protocols. Though such an analysis is a considerable effort, it eliminates the need for expensive domain experts to undergo OO training and modeling sessions. In our approach, each expert spends only about one hour.

In refining the model, we find the recent developments of analysis patterns very useful. We feel that future developments in this area can be of benefit to all OO developers. The CASE tool and methodology (UML) we use provide enough power and flexibility for modeling frameworks. Frameworks for semistructured domains, such as sales promotion, require the use of abstraction techniques like bottom-up design and information hiding, which are supported by CASE tools. We found that creating sequence diagrams is very useful in validating the classes designed.

Reuse is an important theme when building software for domains such as marketing. Although no two product managers have the exact same job, the vast majority of what such a manager does is the same across companies, products, markets, and countries. A sales promotion domain framework can serve as the basis for the development of applications for sales promotion decision making in different companies, markets, and countries, adapted to the needs of specific products, of individual managers, and for the various tasks that have to be performed within the domain. This type of structure probably exists for many such domains, making the development of a domain framework, as we have shown in this chapter, very beneficial.

2.4 Summary

In this chapter we have presented a methodology for developing domain frameworks in semistructured domains, such as marketing. The procedure starts from protocols of experts solving sample problems, after which content analysis is used to derive concepts and relationships between concepts from these texts. In building the OO model, content analysis provides a vehicle for finding classes and relationships that is much

more powerful than those specified by many of the guidelines given in OO literature. Once developed, such a framework can be used for building different applications within the task domain of the user. The illustration of the framework development method presented in this paper concerns decision making about sales promotions.

The work described here can be extended in several directions. For example, the number of sample problems (in this case, two) and the number of experts (20) can be increased. Also, this methodology can be applied to other decision domains with a semistructured character, for example, investment decisions, management development, personnel selection, and legal problem solving.

This chapter has shown that the development of a domain framework, combining methods from behavioral sciences such as content analysis with OO modeling, can make a major contribution to the development of software applications for supporting decision makers in semistructured domains.

2.5 References

[Aamodt-Plaza 1994] Aamodt, A., and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICOM* 7(1), January 1994.

[Blattberg-Neslin 1990] Blattberg, R.C., and S.A. Neslin. *Sales Promotion: Concepts, Methods, and Strategies.* Englewood Cliffs, NJ: Prentice Hall, 1990.

[Booch-Rumbaugh 1997] Booch, G., and J. Rumbaugh. Unified Modeling Language: Metamodel Description Version 1.0. Rational Software Corporation, www.rational.com/uml/.

[Fayad-Cline 1996] Fayad, M., and M.P. Cline. Aspects of software adaptability. *Communications of the ACM,* Theme Issue on Software Patterns, Douglas Schmidt, Mohamed Fayad, and Ralph Johnson, editors, 39(10), October 1996:58—59.

[Fayad-Laitinen 1998] Fayad, M.E., and M. Laitinen. *Transition to Object-Oriented Software Development.* New York: John Wiley & Sons, 1998.

[Fowler 1997] Fowler, M. *Analysis Patterns: Reusable Object Models.* Reading, MA: Addison-Wesley, 1997.

[Gibson-Senn 1989] Gibson, V.R., and J.A. Senn. System structure and software maintenance performance. *Communications of the ACM* 32(3), March 1989:347–358.

[Johnson-Foote 1988] Johnson, R.E., and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming* 2(1), January–February 1988.

[Korson-McGregor 1990] Korson, T.D., and J.D. McGregor. Understanding object-orientation: A unifying paradigm. *Communications of the ACM* 33(9), September 1990: 40–60.

[Krippendorff 1980] Krippendorff, K. *Content Analysis: An Introduction to Its Methodology.* Thousand Oaks, CA: Sage Publications, 1980.

[Rumbaugh 1991] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall, 1991.

[Schank 1982] Schank, R. *Dynamic Memory: A Theory of Learning in Computers and People.* New York: Cambridge University Press, 1982.

[Smolander 1991] Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen, and Pentti Marttiin. MetaEdit—A flexible graphical environment for methodology modeling. *Advanced Information Systems Engineering, Third International Conference CAiSE 1991:*

Proceedings, pp. 168–193, R. Andersen, J.A. Bubenko, Jr., A. Solvberg, editors. Berlin: Springer-Verlag, 1991.

[Taligent 1997] Taligent, I. *Building Object-Oriented Frameworks*. Taligent, www.taligent .com, 1997.

[Weber 1985] Weber, R.P. *Basic Content Analysis*. Beverly Hills, CA: Sage Publications, 1985.

[Wierenga-Van Bruggen 1997] Wierenga, B., and G.H. Van Bruggen. The integration of marketing problem solving modes and marketing management support systems. *Journal of Marketing*, July 1997, pp. 21–37.

[Yourdon 1994] Yourdon, E. *Object-Oriented Systems Design: An Integrated Approach*. Englewood Cliffs, NJ: Prentice Hall, 1994.