Kenneth S. Rubin and Adele Goldberg

Object Behavior Analysis

Analysis is the study and modeling of a given problem domain, within the context of stated goals and objectives. It focuses on *what* a system is supposed to do, rather than *how* it is supposed to do it (which we consider the design aspects). In addition, it must embody the rule of traceability (*why*), which justifies the existence of a given result by tying it back to the stated goals and objectives. The components of the problem domain can be described as anything that end users of the system, both humans and machines, view as part of the problem context. This may include technical issues, if the users view such issues as part of the problem.

We want the analysis process to be carried out in a predictable and controllable manner. In taking an object-oriented approach to analysis, our goal on completion is that we have a clear understanding of the behaviors exhibited by the system, the objects that exhibit these behaviors, the relationships among the objects, and how the objects interact with one another (the system dynamics). This must all be specified in a clear and well-defined language of object and behavior names, chosen from the problem domain. In addition, any implementation code must be traceable back to the results of the analysis. This means that the vocabulary and structures apparent in the design and implementation must clearly reflect the vocabulary and structures that result from the analysis.

Object-Oriented Analysis

Object-oriented analysis endeavors to model a situation in terms of a collection of interacting entities, each of which provides a well-defined set of *behaviors* and *attributes*. Most published approaches describe conceptually similar definitions, although they adopt alternate terminologies [2, 3, 10]. There is a high degree of agreement on the desired structure of the end result; we differ in how to get to the end result.

Many approaches recommend first searching for the tangible objects, notably seeking the nouns in a requirements specification and any applicable verbs and adjectives. With nouns as the objects, the message interface is determined from the verbs, and the logical properties are derived from the adjectives. Although this basic approach may work for small systems, it is our experience that it simply will not scale up. First, it assumes that a complete, formal and correct requirements specification exists. This is almost certainly not true for large systems. In addition, this approach has a strong bias toward the tangible aspects of a problem (i.e., those things that can be seen, heard, felt, smelled, and tasted). Tangible objects are often important to recognize and capture. But, just as often, the conceptual objects have significant influence on the structure of the analysis results. Nouns and verbs are often an insufficient guide to locating these types of objects.

Furthermore, a common claim is that, by adopting an object-oriented point of view, all tangible objects necessarily become part of the analysis result. This is a naive, and often erroneous interpretation. For example, consider the tangible piece of paper in many businesses known as the Purchase Order. Although this paper document exists in many departmental operations, it is often not required as a separate entity in a computer-based system. What is important to understand is the nature of the required information processing, in terms of the services to be performed. Once these services are understood, we can determine what kinds of entities are best suited to carry them out. "Things" well suited to the paper world may not be well suited to the electronic world.

Object Behavior Analysis

We need a more effective way of finding the objects. The approach we use emphasizes first understanding what takes place in the system. These are the system *behaviors*. We next assign these behaviors to parts of the system, and try to understand who *initiates* and who *participates* in these behaviors. These two kinds of players, the initiators and the participants, help us understand the roles of different aspects of a system, and which parts of the system must take responsibility for providing services and managing system information. Initiators and participants that play significant system roles are recognized as objects, and are assigned the behavioral responsibilities for these roles.

This analysis approach is called "Object Behavior Analysis" or OBA. It is a five-step process whose outcome is in the form of:

- Scripts that record the use of the (proposed) system
- Glossaries of:
  - Initiator-participant names
  - Participants' services

  - Attributes
  - State definitions
- Object models
  - Hierarchical relationships
  - Contractual relationships
- System dynamic models
  - Object life cycles
  - Sequencing of operations

OBA is part of a larger process model incorporating the specific engineering opportunities introduced by object-oriented technology [9]. The remainder of this article provides an outline of the specific steps of the OBA methodology. This methodology describes how to conduct the analysis of a problem situation. The outcomes of the analysis must be captured in some notation for purposes of communication. With the special exception of the script notation and glossaries, many of the notations recommended by other published sources [2, 3, 8, 10, 12] are appropriate.

The Order of Analysis and Design

The issue of where to start is at the heart of distinguishing among various analysis approaches. Our goal is to be able to answer the question: Which roles and responsibilities are needed in order to accomplish the required tasks? Furthermore, we must answer why a particular object exists, why it is linked to another object, why a particular service is provided by an object, and how the object participates in fulfilling the functional requirements. Each step of OBA contributes to the explanation or is a source for information used in determining the final outcomes. As such, each of the steps of OBA ultimately must be completed.

OBA can be characterized as an iterative approach, but one with multiple entry points. Within this article, the steps of OBA are stated in a linear manner for purposes of exposition. In practice, they are used both iteratively, within a single part of a project, and in parallel, on multiple parts of a project. When we complete a step, we do so believing we have gathered sufficient information or results for the next step. On moving to the next step, we might discover that information is missing or that new questions are raised. To resolve these issues, we iterate back. The steps of OBA are specifically designed to provide this form of check and balance, to verify that the growing context of the analysis is internally consistent across all steps.

In the previous section, we argued in favor of starting the analysis process with a focus on behavior. In several situations, however, this might not be possible. For example, any one of the following could have been completed prior to the decision to apply OBA:

1. A nonobject-oriented, data-oriented approach has produced a data model.
2. An enterprise-wide analysis has defined a common vocabulary for functions and/or data.
3. A domain analysis has been completed with the basic objects being proposed.

As we explain each step, we will note possible alternative entry points based on having the information from these situations.

The Steps of Object Behavior Analysis

Object Behavior Analysis (OBA) consists of five steps:

- Setting the context for analysis
- Understanding the problem by focusing on behaviors
- Defining objects that exhibit behaviors
- Classifying objects and identifying their relationships
- Modeling system dynamics

The goal of OBA is first, to understand the problem description and, second, to formulate this description in terms of multiple interacting objects. These objects fill system roles and responsibilities by both providing and contracting for well-defined services that carry out system behaviors. The analysis result should be understandable to the end user, lend itself to further design and implementation, and be traceable to system goals and objectives. Based on OBA, estimates can be formulated for the remaining project development based on the numbers of identified behaviors, participants and initiators, and relationships among these various parties.[1]

We label the five steps of OBA as Steps 0 through 4, emphasizing the first step with the unusual label "zero" in order to bring attention to the fact that this step is often outside the scope of what is traditionally called analysis. Figure 1 contains an outline of each of the five steps, their substeps and activities. The structure of the article follows the outlined steps, and we use the labels shown in the figure when identifying substeps. In addition, the article presents an example analysis of an electronic spreadsheet application that illustrates each step.

Step 0—Setting the Analysis Context

Step 0 consists of four substeps that identify goals and objectives, appropriate resources for analysis, core activity areas, and a preliminary analysis plan.[2] Carrying out these substeps forms the foundation for the context in which we carry out analysis. In particular, Substep 0.1 identifies goals and objectives, which are statements of the desired system outcome. *Business goals* identify the specific business reasons for building a system. As analysts, we do not reason about the appropriateness of these goals, but require them as the base on which we can measure our progress and success. Where possible, a list of system features should also be iden-

[1]We are currently developing a set of metrics based on the number of system behaviors (as opposed to the number of lines of code "kloc"). Our work is influenced by the work on Function Point Analysis [6].

[2]The authors would like to acknowledge the influence of Gurdon Blackwell of Gemini Consulting, Morristown, N.J., in formulating this all-important zeroth step.

tified. Objectives differ from goals in that they are time-targeted, measurable descriptions of every key aspect of the project. There are several categories of objectives, notably resource and quality. *Resource objectives* define the people, time and money budgeted for the desired project. *Quality objectives* are quantitative descriptions of qualitative results. Examples are performance, reliability, and reusability. We use Gilb's quality templates [4] in specifying quality objectives, one example of which is shown in Figure 2.

Substep 0.2 serves to identify resources—documents, dictionaries, as well as end-user and domain experts—that can contribute to the analysis effort. The next substep, Substep 0.3, is to identify the core activity areas. These are the major areas of the system that require analysis. Identifying these areas will provide a basis for the scripting process (discussed in Step 1) and for work partitioning and parallel development. One way to identify these areas is to sketch out the *pristine life cycle* of the system. This is typically a time-sequenced ordering of the major activities that occur in a problem domain.

Another way of determining core activity areas is to examine the operation of a current system (electronic or not), and determine the major clusters of behavior. And a third is to make a guess or use naive knowledge of the system problem to identify key high-level tasks that should be carried out. There is no requirement that the first-pass core activity areas survive over time; the final version is likely to have evolved over the life of the analysis. However, it is useful to generate an initial set to help begin the process.

The last task, Substep 0.4, is to generate a preliminary analysis plan. The plan takes the partitioning of core activity areas, and establishes priorities and makes estimates of time and resources for analysis activities. This plan is integrated into the master project plan.

For the purposes of our example, suppose we run an accounting department in a small software company that requires automated support tools. Although several popular electronic spreadsheets are commercially available, we feel that their prices are prohibitive. So we decide to create the necessary tool ourselves. The engineering group is asked to create a limited-functionality spreadsheet application. Figure 2 contains the general description and constraints given to the engineering group, as well as results from Step 0 of OBA.

Figure 1. Outline of OBA methodology

- Step 0 – Setting the Analysis Context
  - Substep 0.1 – Identify goals and objectives
  - Substep 0.2 – Identify appropriate resources for analysis
  - Substep 0.3 – Identify core activity areas
  - Substep 0.4 – Generate preliminary analysis plan
- Step 1 – Understand the Problem
  - Substep 1.1 – Scenario Planning
    - Choose major scenarios
    - Map scenarios to core activity areas
  - Substep 1.2 – Scripting
    - Define script metadata
    - Specify each step as initiator-action-participant triplet
    - Determine participant service
    - Determine scripting issues
    - Update scripting issues table
    - Divide scripts
  - Substep 1.3 – Build Glossaries
    - Generate Parties Glossary
    - Generate Services Glossary
  - Substep 1.4 – Deriving Attributes
    - Examine scripts for initiator and participant attributes
    - Generate Attributes Glossaries
- Step 2 – Defining Objects
  - Substep 2.1 – Generate modeling cards
    - Determine different types of objects
    - Accumulate attributes
    - Accumulate provided services
    - Accumulate contracted services
- Step 3 – Classifying Objects and Identifying Relationships
  - Substep 3.1 – Describe contract relationships
  - Substep 3.2 – Organize objects into hierarchies
    - Choose organizing principle(s)
    - Determine abstractions
    - Determine specializations
    - Factor objects
    - Generate/update reorganization table
- Step 4 – Modeling System Dynamics
  - Substep 4.1 – Generate State Definition Glossaries
    - Determie states associated with each object
    - Define each state
  - Substep 4.2 – Determine object life cycle
    - Identify events
    - Organize into life cycle
  - Substep 4.3 – Determine sequencing of operations

Step 1—Understand the Problem

Once the context has been set, the

next step is to determine what the system is supposed to do, and for whom and with whom it is supposed to do it. The basic idea is to specify use scenarios that cover all possible pathways through the system functions. (Jacobson's Objectory approach also suggests a similar technique that he calls Use Cases [7].)

One approach to obtaining the use scenarios is through a structured interviewing process.[3] Two kinds of people are typically interviewed: the users and the domain experts. Users are the people who perform some of the activities in the current system, or who will perform activities in the system to be built. Experts vary, depending on the type of system. Generally, experts include the people who have sponsored the work and who have expectations about the system, consultants who work in the domain and are considered knowledgeable, as well as other developers who are experts in building systems of the same type.

In situations in which interviewing is either not possible or not desired, the analyst might rely on existing written documentation or personal knowledge in order to construct the use scenarios. These approaches are especially useful when the problem is an engineering one, such as creating an electronic spreadsheet. In the example of the spreadsheet application, we identify several example spreadsheets that we wish to construct. The use scenarios describe in detail how the construction of these example spreadsheets is carried out. One such example is provided in Table 1. In Substep 1.2, we use *scripting* and a special script notation for capturing use scenarios. This notation is a simple tabular form as shown in Table 2.

The basic idea behind a script is directly related to the operational concept of an object-oriented system. Specifically, there are a collection of entities in the system, each of which provides a set of well-defined services that may be used by other entities. Work gets done when one entity communicates with another to notify that an event has taken place, to provide information, to request information, or to request service. Scripts are designed to capture this information in the context of a particular use scenario that defines a sequence of service requests in order to accomplish some overall task.

In order to understand scripts, we must define our terminology. We say that a *contract* is an agreement between two entities such that one entity will utilize (invoke) a service provided by the other. This definition is similar, but not identical, to other definitions of contract in the literature [13]. Two parties are involved in any contract: the

Substep 0.1 Goals and objectives

Business Goals

Support the accounting department budget preparation with appropriate tools by creating a simple spreadsheet manager whose features are a subset of those available on the personal computing market.

System Features

Presented as lined grid with delineated border
Label rows of cells from 1 to 16384
Label columns of cells from A to IZ
Cell can hold arbitrarily large strings, integers, floating-point numbers, percentages, dollars, and expressions
Expressions cannot contain strings
Currency in expressions are treated as floating point for purpose of arithmetic evaluation
Expressions can not contain cycles
Evaluation of the cells and redisplay is continuous
Main commands are: creating new spreadsheet, retrieving one from file, formatting, file management, and printing
It is possible to select an individual cell or arbitrary two-dimensional collection of cells

Resource Objectives

Personnel: Donna
Time: 3 person-months

Quality Objective for Response Time

(this would be one of several quality objectives)

scale	seconds to recalculate whole spreadsheet
prerequisite	expressions exist in several cells, and at least one value changed
test	carry out calculations for any change in all specified examples
worst	30 seconds
plan	20 seconds
best	immediate

Substep 0.2 Analysis resources

User: Accounting Manager, Adam
References: Excel and Lotus 1-2-3 user manuals

Substep 0.3 Core activity areas

- Creation
- Modification
- Calculation
- Save and Load

Substep 0.4 Preliminary analysis plan

- No users have to be interviewed
- Use a Salary Plan Example (one example is shown in Table 1)
- Script each activity area by creating an example to construct
  - Script a path with no errors
  - Opening and closing a spreadsheet should include use of files
- Script special situations
  - Note possible input errors that could occur
  - Note special external events that could occur

Figure 2. A simple spreadsheet—Step 0 of OBA

[3]There are several good references on structured interviewing techniques, notably from cognitive psychology and the artificial intelligence communities [11].

initiator and the participant. The *initiator* is the party responsible for invoking the service of another party. The *participant* is the party that provides the service. An *action* is a behavior of an initiator that causes a service invocation on the part of the participant. A *service* is a behavior of the participant that can be contracted for use by another party. In other words, it is a behavior that can be invoked via the available service interface of the participant. In this article, unqualified uses of the word *service* always mean *participant service,* which can be described as a participant's response to the question: "What services do you provide to other parties?" *Contracted services* are an initiator's response to the question: "What *services* do you contract to use in order to carry out your behaviors?" The actions of an initiator indicate its contracted services.

The columns of a script are labeled with the four key terms: initiator, action, participant, and service. Each row indicates a contract between an initiator and a participant. Using Table 2 as the example, we can interpret each row as follows. In the first row we see that thing1 (initiator) notifies (action) thing2 (participant). We interpret this to mean that in the execution of the use scenario, thing1 wants to interact with thing2 and, in particular, it wants to notify thing2 that something has taken place in the system. Since we are developing an object-oriented model of the problem, the only way thing2 can be notified is if it provides some service that allows it to be notified. This is denoted in the fourth column of the script as the participant's service. So, thing1 is contracting to use a service of thing2, and we see this contract invoked explicitly by the corresponding action of thing1.

A service is not a reaction of the participant to an initiator's action. Rather, it is a specification of what interface the participant must provide in order for the contract to be fulfilled. Any reaction on the part of the participant would normally be the next line of the script. This next line might have, for example, the roles of the initiator and the participant swapped. The remaining rows in Table 2 illustrate the basic forms of contracts that may exist between an initiator and a participant.

Table 3 contains a script for the spreadsheet example. The first action listed indicates that a User (initiator) selects the D1 cell of the Spreadsheet (participant). Notice the content of the service column: select a cell. In order for the User to select D1 from the Spreadsheet, it must be the case (in an object-oriented system) that the Spreadsheet provides a service to select a cell. Services are expressed in a brief declarative format.

Each script contains other important information, illustrated in Table 3, including:

- Name or Identifier
- Author
- Interviewee(s) or References, if appropriate
- Version or other change history
- Preconditions
- Postconditions
- Trace to a goal, objective, core

Table 1.
A Simple Spreadsheet—Step 1.1 of OBA

	A	B	C	D
1				NEW
2	NAME	SALARY	%RAISE	SALARY
3	Joe	$55,000	4%	$57,200
4	Mary	$60,000	4%	$62,400
5	Henry	$30,000	4%	$31,200
6	Abel	$25,000	4.5%	$26,125
7	Sam	$40,000	4%	$41,600
8	Jane	$30,000	4%	$31,200
9	Betty	$25,000	5%	$26,250
10	Martha	$30,500	4%	$31,720
11				
12	TOTALS	$295,500		$307,695
13	Average Raise		4.2%	$1,524
14	Expense Increase			$12,195

Table 2.
Script Notation

Initiator	Action	Participant	Service
thing1	notifies	thing2	thing2 can be notified
thing1	provides info to	thing2	thing2 can accept info
thing1	requests info from	thing2	thing2 can provide info
thing1	requests service from	thing2	thing2 can provide service

activity, or another script

The name uniquely identifies the script. The preconditions denote what must be true in order for the script to be applicable. The postconditions denote what is true of the world as a result of carrying out the script to completion. This information is helpful later in generating the full system dynamic model and in determining relationships among the scripts. Pre- and postconditions are expressed in terms of state descriptions and objects. For example, we would express the notion that an account in a banking system is overdrawn simply as "overdrawn (Bank Account)."

A number of rules, not detailed in this article, apply when scripting. (At this point they are only documented in course notes [9]). Most important, however, scripts must be understandable to the interviewees and other analysts.

Actions in scripts can be marked to indicate four possible considerations. They are shown here with the marks we have adopted. Each mark in the script is uniquely numbered to reference a note with additional explanation. Such notes are maintained in various tables, in particular, a Design Issues Table, Analysis Issues Table, and an External Issues Table.

- **A?** Additional information needed in order to complete analysis
- **A >** Explained in another script
- **D** Elaborate at design time
- **E** Outside or external to scope of system

All issues flagged with A? or A > must be resolved before the analysis phase is considered complete; issues considered to be outside the scope of the system should come under scrutiny during the analysis review.[4] All issues flagged with D must be resolved during the design phase. Once a script is completed, interviewees or other resources are consulted to make sure all terms are meaningful, and the level of detail is appropriate to the resources involved.

Table 3.
A Simple Spreadsheet—Step 1.2 of OBA

Script Name	Modification.1.example
Author	Donna
Version	1.0
Precondition	exists (Spreadsheet), displayed (Spreadsheet)
Postcondition	modified (Spreadsheet)
Trace	Core Activity—Modification

Initiator	Action	Participant	Service
User	select D1	Spreadsheet	select a cell
User	type text NEW	D1	set content to text
User	set text style to bold	D1	set text style to bold
User	select A2	Spreadsheet	select a cell
User	type text NAME	A2	set content to text
	(repeated select and type text for example)	B2, C2, D2, A3 through A10	
User	select Row 2	Spreadsheet	select a row
User	set text style to bold	Row 2	set text style to bold
User	extend row height to 34 pixels	Row 2	resize height
User	select A12	Spreadsheet	select a cell
User	type text TOTALS	A12	set content to text
	(repeat select and type)	A13, A14	
User	select A12:A14	Spreadsheet	select vertical collection of cells
User	set text style to bold	A12:A14	set text style to bold
User	select B3	Spreadsheet	select a cell
User	type number 55000	B3	set content to number
User	choose format $xx,xxx	B3	set format to currency
User	select B3:B10	Spreadsheet	select vertical collection of cells
User	copy first cell into rest of cells	B3:B10	fill down
User	select B4	Spreadsheet	select a cell
User	replace 55 by 60	B4	set content to text
	(repeat rest of changes)	B5:B10	
User	select Column A	Spreadsheet	select a column
User	contract column width to 30 pixels	Column A	resize width

Many scripts are written when carrying out a complete analysis. Creating scripts introduces potentially new terminology. In Substep

1.3, we create a set of glossaries in which each new term is defined. Three specific glossaries are created:

- Parties
- Services
- Attributes

Tables 4a and 4b contain versions of an example parties glossary. The first version is directly derived from the script shown in Table 3. The format for the parties glossary defines each initiator or participant name in terms of the role each party plays in the system. In addition to the role definition, each entry for a party includes traces back to the scripts in which the party is referenced. Associated with each trace is an indication as to whether or not the party played the role of an initiator (I), participant (P), or both (I/P).

Table 4a.
A Simple Spreadsheet—Step 1.3 of OBA

Parties Glossary—Version 1

Party	Definition	Traces	Role
User		Modification.1.example	I
Spreadsheet		Modification.1.example	P
D1		Modification.1.example	P
A2		Modification.1.example	P
A12		Modification.1.example	P
Row 1		Modification.1.example	P
A12:A14		Modification.1.example	P
B3		Modification.1.example	P
B3:B10		Modification.1.example	P
B4		Modification.1.example	P
B5:B10		Modification.1.example	P
Column A		Modification.1.example	P
D1		Modification.1.example	P
Row 2		Modification.1.example	P
Column A		Modification.1.example	P

Table 4b.
Parties Glossary—Version 2

Party	Definition	Traces	Role
User	human or other driver of the application	Modification.1. example	I
Spreadsheet	2D grid of cells that can contain data and formula for computing (in a noncircular manner)	Modification.1. example	P
Cell	container for an element that has a value	Modification.1. example	P
Vertical cell collection	contiguous set of cells from the same column	Modification.1. example	P
Row	collection of cells occupying the full horizontal dimension of the grid	Modification.1. example	P
Column	collection of cells occupying the full vertical dimension of the grid	Modification.1. example	P

This example is interesting in the sense that the scenario scripted was very specific in identifying a particular cell (A2), a particular row (Row 2), a particular column (Column A), or a particular collection of cells (A12:A14). The problem statement, including the list of features, gave more general reference names we can use to simplify the glossary. We can therefore choose general party names for the specific participants—Cell, Row, Column, Vertical Cell Collection. These are shown in Table 4b.

Consistent with our requirement to maintain traceability, we must keep track of these name generalizations. We do this in an Alias Table consisting of a General or Preferred Name and its Aliases, as shown in Table 4c. The services glossary, shown in Table 5, contains a definition for each service name, based on the script action from which the service is derived. Each service entry includes the list of participants who exhibit the behavior and traces to all the scripts in which the service was identified.

The final task of Step 1, Substep 1.4, is to derive the attributes of each party in the script, and describe these attributes along a number of dimensions. An attribute is a logical property of a party that is associated with the requirement to fulfill one or more contracts. If it is associated with the initiator, we can assume the initiator requires the attribute in order to invoke the service. Similarly, if it is associated with the participant, we can assume the participant requires it in order to fulfill the service.

As stated, attributes are logical and not necessarily physical properties. This distinction is made clear by the following example. From a script, we might conclude that an attribute of a person is age and that this person can be asked, "How old are you?" The assumption that the person can answer this question does not imply that age is a physical attribute (i.e., a stored value). Perhaps the person stores a birth date and computes age by taking the difference between the current date and the birth date. The decision as to how a logical attribute is physically realized is a design issue. The relationship between the party and its attribute, however, is an analysis issue. Knowing the semantics of this relationship will provide the designer with information needed to make decisions regarding a physical implementation.

A common practice at analysis time is to draw some form of semantic net diagram that represents parties and attributes as nodes, and the relationships among them as named arcs. Many of the published notations make special provisions for showing composition (a.k.a. whole-part) and cardinality [2, 3, 10]. The benefit of doing this is to provide a graphical perspective of parties and their attributes, a perspective that can be effective at communicating the relationships.

The graphical diagrams are useful, but not sufficient in the sense that it is difficult to communicate the total essence of the relationship between a party and its attribute using a single word descriptor. In the past, we tried to define a small set of relationships between objects and their attributes—names such as: "knows-about," "communicates-with," and "has-as-part." We concluded that no such small set exists that adds real value to capturing the deep semantic relationships and at the same time can be used by the designers to specify the deliverable system. Our current approach is to capture dimensions of the relationship in an extensible table format, the attribute glossaries, as illustrated in Table 6. When desired, we can augment these with diagrams that, in fact, can be generated from the glossaries.

There is an attribute glossary for each party that has identified attributes. We require separate glossaries because attributes are private to each party, and common vocabulary at this level is not meaningful. Table 7 describes the purpose of individual columns, in particular, to capture the relationship between the attribute and service contracts. Note that the rows of this table are interpreted by designers to make decisions about the physical structure of the system.

Table 4c. Alias Table

General or Preferred Name	Aliases
Cell	A2, A12, B3, B4, D1
Vertical Cell Collection	A12:A14, B5:B10
Row	Row 1, Row 2
Column	Column A

In each case, rather than saying simply "yes," more specific information should be provided. For example, which contract, the name of the accessor or mutator service, or the cardinality or constraint of the type of collection. Attributes for only one party identified in the example script of Table 3 are shown in Table 6.

As the process of scripting continues, it is important to have the interviewees and analysts work together to agree on the semantics for each of the entries in the three kinds of glossaries. This assists everyone in understanding the problem domain. All scripts must use the same name when referring to the same action, initiator, participant, or service, or a defined alias. Skilled analysts act as facilitators to win consensus on these definitions. The purpose of obtaining this consensus is to determine a unique and persistent description for each aspect of the system. Performing this normalization of vocabulary prospectively gives a first-pass enforcement of polymorphic protocols (that is, it identifies the common message names to be shared by different objects).[5]

It is possible that a data-oriented analysis approach or an enterprise-wide analysis has already been carried out before OBA is started. In such cases, it is likely that a vocabulary for any aspect of the analysis—initiators, participants, services, or attributes—has already been determined. The challenge then is to start with the glossaries, fill them out based on the prior non-OBA analysis, and then devise scripts that support the decision to include each aspect.

Step 2—Define Objects

Up to this point, we have created use scenarios that describe core activities, and whose content is based on interviews, example constructions, or references (experts or written materials). We have captured these scenarios of how things should work, and presented them in the form of scripts. The scripts are linked together by matching postconditions with preconditions, giving a larger picture of how actions might progress in the system. In doing so, we identify a number of parties that act as either initiators, participants or both. In addition, we identify the contracts and the required participant services, as well as the logical properties of the

[4]We assume that analysis is carried out in the context of a project process model which includes a system of periodic reviews.

[5]In other words, we are obtaining a common vocabulary incrementally and iteratively as part of working with the end users to obtain understandable scripts. The incremental building of the glossaries provides multiple interviewers with the basis for shared terminology with end users, rather than making up new words that then need to be renegotiated. This avoids the need for a post-normalization pass over the results of the scripting to clean up terminology and arrive at a common vocabulary. This yields a set of objects with polymorphism already specified.

parties needed to invoke or carry out services. We are now ready to select the parties that should be analysis objects.[6]

To do so, we first note that initiators that are not participants will also not be objects. This is a consequence of the very definition of objects—that objects are in part defined by a well-defined service interface. Parties that are not participants have no service interface. Typically these initiators reside just outside the scope of the current system, or right on the boundary. They are interesting in the context of the analysis because they contract to use the services of objects inside the scope of the system. To this end, they help identify the services of these objects.

Any party that provides a service is a potential object. There are several cases to consider:

1. Participants that are not initiators. These are usually data-store objects that provide behavior for accessing and mutating stored values.

2. Participants that name a collection of objects. For example, in an automatic bank teller domain, the participant Bank System may really be the name of a collection of finer-grain objects. As such, Bank System itself may not be an object in the system, but rather the name of an aggregation of objects.

3. Participants that are also initiators and do not appear to be overburdened with respect to their roles/responsibilities. These will be the most abundant objects. Note that we use the notion of overburdening to raise the question as to whether the participant may really be naming several objects.

Whether or not a participant is

[6]We call these "analysis objects" to raise the issue that in fact these are the objects that directly map to the problem space. Not all of these objects will necessarily remain once the design is completed; moreover, new objects will typically be introduced during design to support particular architecture and implementation decisions.

Table 5.
A Simple Spreadsheet—Step 1.3 of OBA

Services Glossary

Service	Definition	Participants	Traces
select a cell	select a single cell and make it the current selection	Spreadsheet	Modification.1. example
set content to text	set the contents of a cell to be a Text	Cell	Modification.1. example
set content to number	set the contents of a cell to be a number such as a Float or Integer	Cell	Modification.1. example
select a row	select a row of cells and make it the current selection	Spreadsheet	Modification.1. example
set text style to bold	set the text style to a bold emphasis	Cell	Modification.1. example
		Row	Modification.1. example
		Vertical Cell Collection	Modification.1. example
resize height	change the height of a given row	Row	Modification.1. example
select a vertical collection of cells	select a partial column of cells and make it the curent selection	Spreadsheet	Modification.1. example
set format to currency	set the format as Currency	Cell	Modification.1. example
fill down	replace remaining contents of each cell in vertical selection with replication of first cell in the selection	Vertical Cell Collection	Modification.1. example
select a column	select a column of cells and make it the current selection	Spreadsheet	Modification.1. example
resize width	change the width of a given column	Column	Modification.1. example

Table 6.
A Simple Spreadsheet—Step 1.4 of OBA

Attribute Glossary—Row

Name	Definition	Contract	Accessor	Mutator	Multi/Single Value	Range of Values	State Defn
height	height of the row in number of pixels	none	none	resize row height	single	integer 24 .. 1024	no
format	interpretation of the contents	none	none	set format	single	$xx,xxx	no
style	presentation of contents	none	none	set style	single	bold	no

overburdened is a question of the number of roles the participant is playing as reflected in the diversity of services. In the spreadsheet example, we might imagine that the spreadsheet object is given responsibility for managing the 2D array of cells and, in addition, manages the format and style of each of the cells. Our scenario steered us to assign responsibility for format and style to the cell. If we had written the scenario differently, however, we might have found ourselves with an overburdened spreadsheet and, at this step, recognized the problem. We would then iterate back to Step 1 in order to determine an appropriate change. There are no concrete rules other than to make sure that services of an object relate reasonably to one another and to the intended role of the object.

To begin organizing the information we gathered in Step 1, we have adapted the idea of CRC cards as a technique for capturing information related to a proposed object [1, 13]. However, we have expanded the information content of the cards, and set up prior steps so that the initial cards can be fully generated from information contained in the glossaries. We call these Object Modeling Cards. As shown in Figure 3, they contain:

- Name of the object
- Names of the objects from which attributes and behaviors are inherited
- Information and behaviors added by the object
- Attributes identified with the object
- Services provided by the object
- Services contracted by the object
- Card trace

All names in the Object Modeling Card must agree with the names in the glossaries. So far, in Steps 0–2, we have specified inheritance of neither attributes nor behaviors. This will be done in Step 3. When the party acts as a participant, we list the services provided. If the party also serves as an initiator, we capture the contracted services (i.e., those services the party expects to be fulfilled by others). Information about contracted services is logically done in Step 3, where we identify object relationships.

There are four traces on the Object Modeling Card. The first references the script in which each attribute was identified. The second references a script action in which an initiator invoked a particular service from this object. And the third trace references the script in which the object, as initiator, requested action of another participant, and thereby specified a service to be contracted.

Fourth, we need a trace for the card itself. By default, this trace is blank to indicate that the card was initiated by the analyst as a way to summarize already obtained information. A change to this fourth trace occurs only in Step 3. In addition to the four traces, there is information about the Object Modeling Cards that has to be retained, for example, versioning information.

As noted in prior steps, it is possible that a prior analysis was carried out before OBA, and that the outcomes from this analysis are to be utilized. One possible situation is that a domain analysis has been completed with the basic objects already proposed. In this case, the

Table 7.
Description of Columns in an Attribute Glossary

Column	Description
Name	A unique name for this attribute in the context of the party.
Definition	An unambiguous definition of the attribute in the context of the containing party.
Contract	Does the party have any contracts with the attribute?
Accessor	Does the party provide a service for accessing the attribute?
Mutator	Does the party provide a service for mutating the attribute?
Multi/Single Value	Does the attribute denote a collection of values in the context of the party?
State-Definition	Is the attribute used to define states of the party?

entry point for OBA analysis can be Step 2, to create Object Modeling Cards for the (already) proposed objects. Then it is necessary to iterate back to Step 1, to create scripts that support the proposal and to create the glossaries. It might be possible to reuse the scripts and glossaries from the domain analysis if it were conducted using OBA. In either case, by iterating back it is possible that some of the proposed objects will be eliminated and others added.

Figure 3. A simple spreadsheet—Step 2 of OBA

Figure 4. A simple spreadsheet—Step 3.1 of OBA

Step 3—Classify Objects and Identify Relationships

The tasks in Step 3 involve applying a set of techniques to identify relationships among objects. Applying these techniques enables us to fill in the blanks left over from Step 2 in order to complete the Object Modeling Cards:

- Contracted Services
- Card Trace
- Inherits From

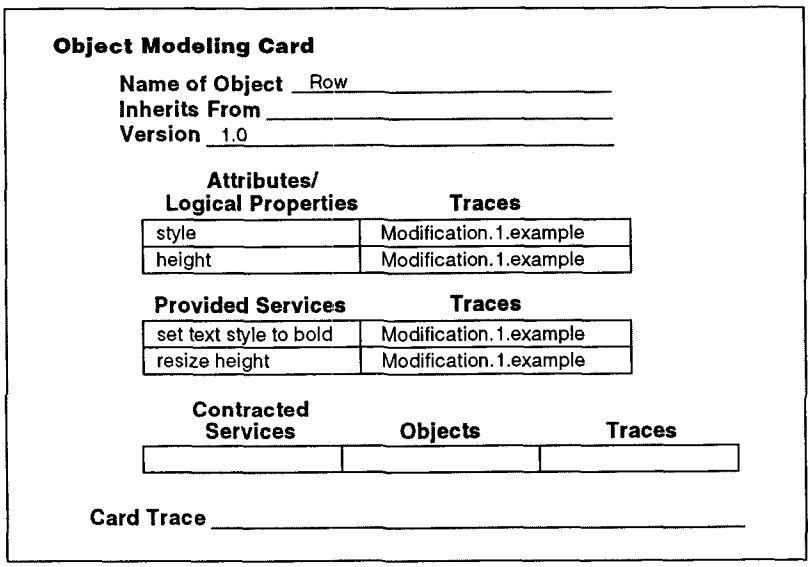
The purpose of Substep 3.1 is to describe the contractual relationships among the objects. Inclusion of contracts in the Object Modeling Card serves two purposes. First, it allows us to derive the relationships between this object and other objects in the system. In object-oriented terms, this contractual relationship is essentially a statement that the object sends a message to another object for the purposes of obtaining information, providing information, requesting action, or notifying that some event has occurred. The second purpose is to avoid errors that result when one object expects a service of another, but no object has taken responsibility for that service.

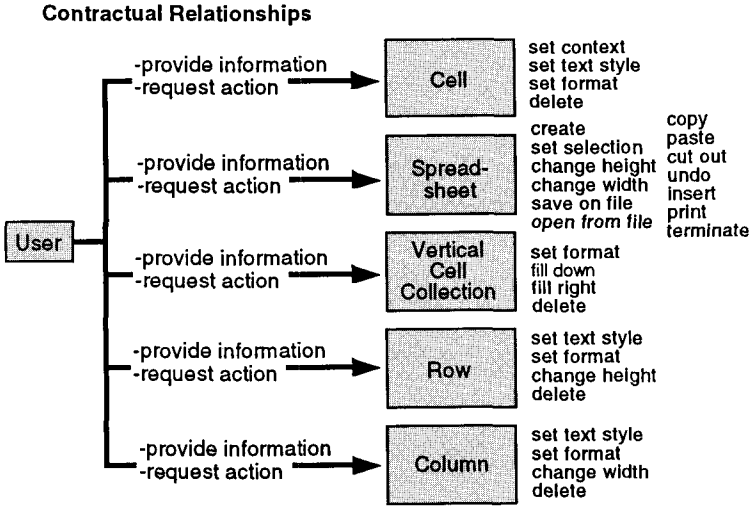
For each Object Modeling Card created in Step 2, we capture the contracts the object expects to be fulfilled by others. An illustration of contractual relationships, partially derived from Object Modeling Cards for the spreadsheet example, are shown in the diagram of Figure 4. In order to provide this and subsequent examples, we assume that more scripts have been created, and a more complete set of objects has been identified.

We are now ready for Substep 3.2 in which we apply several techniques we refer to as *reorganization techniques*. The goal is to determine:

- services common to two or more objects, and to create an object that captures the shared description of these services
- logical properties common to two or more objects (by examination of attribute glossaries) and, again, to create an object that captures the shared description of these attributes
- services or logical properties of one object can be described as a refinement of the services or logical properties of another object
- an object assigned multiple responsibilities (in terms of its provided services), and to factor these into a separate object for each area of responsibility

The first two techniques are called *abstraction*, the third *specialization*,





and the fourth is *factorization.* As an outcome of applying these techniques, we create new objects and their associated modeling cards.

We must maintain traceability despite the introduction of new objects by the application of these techniques—objects that do not appear in the scripts. In order to keep track of the rationale for any change, we create a special reorganization table, as shown in Table 8. Each entry lists the type of technique, which existing objects were inputs to the technique, and which resulted as outputs. The trace contains a justification for the reorganization. If the decision comes under reconsideration at a later date, capturing why it was made can help in avoiding change errors.

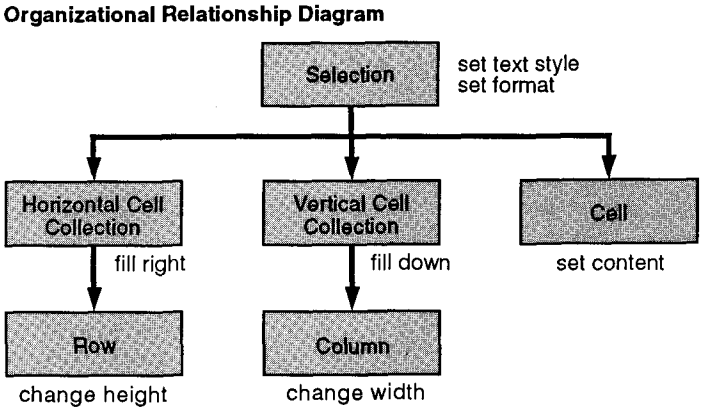
Reorganization can create new objects. New Object Modeling Cards must therefore be created. The Card Trace on any new card indicates how it was created by referencing the appropriate entry in the table of Reorganization Techniques. If the new object is a specialization of another object so that it inherits its services from this other object, then this relationship is recorded in the "Inherits From" field of the Object Modeling Card. Similarly, any new version of an existing object, which recasts itself as inheriting from a new object, will record the inheritance relationship in this field of the new version of its Object Modeling Card.

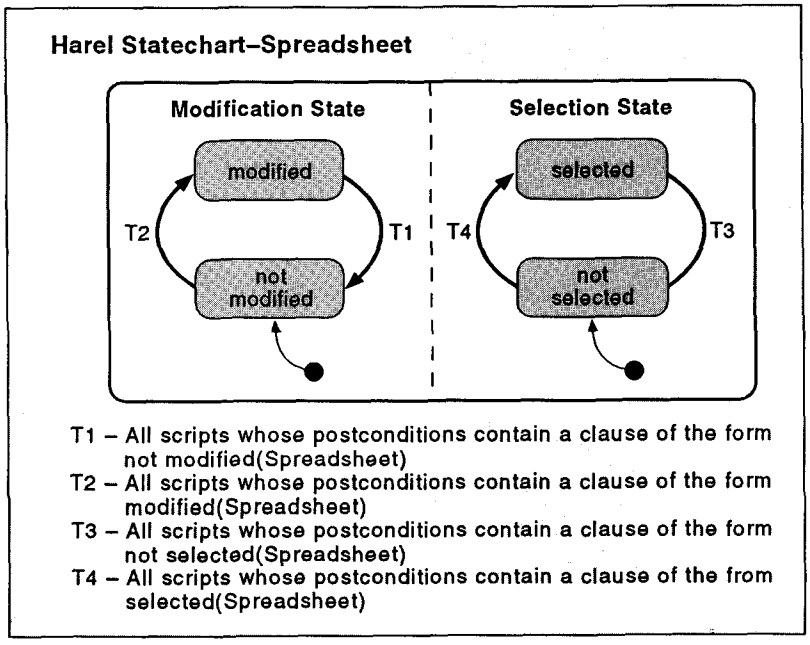
We continue with the spreadsheet example, in which we identified objects we named Cell, Row, and Vertical Cell Collection. From Figure 4, we see that Column and Vertical Cell Collection both share services to delete and to set the format of the selected cells. Based on the geometry of 2D grids, we understand a Column to be a special case of a Vertical Cell Collection; a Column consists of all the cells in the full height of the spreadsheet. Thus we can specify that a Column is a specialization of a Vertical Cell Collection, and allow Column to inherit the delete and set format services. We create a new Object Modeling Card for Column indicating that it inherits from Vertical Cell Collection. Two services specified in Vertical Cell Collection, set format and delete, are inherited by Column. We note that one other service, fill down, is now a service of a Column, and that the Column service—change width—can reasonably be provided as a service of Vertical Cell Collection as well.

Imagine that after further scripting (not shown) we had also defined a Horizontal Cell Collection. Similarly, then, we determine that a Row is a specialization of a Horizontal Cell Collection. Next, we notice that a Spreadsheet provides the service for setting a selection, which could be any one of a Cell, a Row, a Column, or more generally, any Vertical or Horizontal Cell Collection. Thus, it is possible to relate the descriptions of these objects to the more abstract notion of a Selection. It is possible to set the text style or format of any Selection. Again, Object Modeling Cards for new objects or new versions of objects are created, clearly specifying these relationships and assignment

Figure 5. A simple spreadsheet—Step 3.2 of OBA

Figure 6. A simple spreadsheet—Step 4.2 of OBA





of service responsibilities.

Now that we have a collection of Object Modeling Cards, we might wish to see the relationships in a graphical form. Various object relationship diagramming notations, such as those recommended by Booch or Rumbaugh et al., can be used at this point [2, 9]. Figure 5 presents a diagram of the objects and their relationships, representing the outcome of Substep 3.2.

Step 4—Model System Life Cycles

Up to this point, we have dealt with static views of the system we are analyzing. These identify the structure of the system at a single point in time, that is, what behaviors the system contains, which objects are responsible for these behaviors, and any relationships among objects.

Step 4 of OBA is concerned with modeling system dynamics, that is, those aspects of the system that change over time. The system will carry out behaviors in response to events, in a prescribed order. Object states, events, and the order in which behaviors occur must be clearly represented.

The states associated with an object are defined in Substep 4.1. States are used to represent a situation or condition of an object during which certain physical laws, rules, and policies apply (this definition comes from [12]). Changes in state typically result in changes in the behavior of one or more objects in the system. Suppose, for example, that our scripting indicates that there is an application for loading and saving spreadsheets. Whenever the user tries to exit, the application determines whether or not the spreadsheet was modified since it was last saved and, if so, offers to save before exiting. Thus, the state of the spreadsheet—modified or not—affects the behavior of the application.

The states of objects are determined from script pre- and postcondition expressions. Expressions are composed of a collection of clauses, each of which is in turn composed from a state description and object pair. The first pass of determining the interesting states of an object is to search all pre- and postcondition expressions for clauses that contain the object. The state definition is not complete until all such clauses have been considered. Conversely, if we have knowledge of the problem domain that indicates a state condition has not been accounted for, we have evidence to believe that scripting has not been completed, and we should iterate back to Step 1.

Notice that in Table 3, the original script for creating part of the example spreadsheet, we identified the postcondition to be modified (Spreadsheet). This was our first hint that the spreadsheet has a state condition—modified or not—that could affect its behavior or that of other objects.

Each state of an object is defined in terms of a Boolean function over attributes and values. Suppose a Spreadsheet has the state condition "has selection." Further suppose the Spreadsheet has an attribute called current selection. The state condition, then, is defined to be "current selection not empty."

An object can exist in a set of nonoverlapping states. For example, it makes sense to say a Spreadsheet is modified or not, and has a selection or not (we assume that a selection alone does not constitute modification). These states are nonoverlapping in the sense that either state can change without necessarily affecting the other.

A State Definition Glossary, as

Table 8.
A Simple Spreadsheet—Step 3.2 of OBA

Reorganization Techniques

ID	Inputs	Outputs	Traces	Type of Technique
1	Column, ver 1 Vertical Cell Collection, ver 1	Column, ver 2 Vertical Cell Collection, ver 1	Column is a kind of Vertical Cell Collection	Specialization
2	Row, ver 1	Row, ver 2 Horizontal Cell collection, ver 1	Row is a kind of Horizontal Cell Collection	Specialization
3	Cell, ver 1 Vertical Cell Collection, ver 1 Horizontal Cell Collection, ver 1	Selection, ver 1 Cell, ver 2 Vertical Cell Collection, ver 2 Horizontal Cell Collection, ver 2	Selection holds common services for Cell or a Cell Collection	Abstraction

Table 9.
A Simple Spreadsheet—Step 4.1 of OBA

State Definition Glossary

State	Definition	Description	Traces
has selection	current selection not equal to nil	used to indicate when a selection has been made	modification.9 .example precondition (script not shown)
modified	for all cells c, there exists at least one c such that the c is modified	used to indicate that some aspect of the spreadsheet has been changed since the last save	modification.9 .example postcondition

shown in Table 9 for the Spreadsheet, is created for each object that undergoes state changes that affect its behavior. Each glossary contains the name of a given state, its associated definition and description, as well as trace information.

In Substep 4.2, we determine the life cycle of each object for which we created a State Definition Glossary. The life cycle describes how an object moves from state to state in response to events. In an object-oriented system model, an event conceptually occurs any time one object invokes a service in another object. This conceptual view of an event is too fine a level of granularity to help us model and eventually construct a workable system. From this point of view, we would conclude that even the most trivial service invocation causes one or more objects to experience state changes. Since this is not true in practice, we prefer to view an event as an occurrence or change in the system or environment that causes one or more objects to experience a state change that consequently affects the behavior of the system.

So how do we find events? The answer lies with the scripts. Scripts are groupings of activities that can be viewed as singular events. More important, we know that the invocation of a script will cause one or more objects to move from one state to another as defined by the pre- and postconditions of the script. Using scripts as events, and pre- and postcondition clauses as state definitions, we are able to construct the life cycle of a given object. Because an object may simultaneously exist in more than one state, we, like [10], have opted to use Harel's Statechart notation [5] as a means of describing the life cycle. Figure 6 contains an example use of Harel's Statechart.

In constructing an object's life cycle, we may determine that a meaningful state is not currently represented. As mentioned earlier, this is an indication to return to the scripting process. However, we may not have to generate new scripts but perhaps we need to divide existing scripts. This would occur if an object enters an interesting state in the middle of a script. In order to make this state explicit, we divide the script and capture the interesting state in the postcondition of one script and the precondition of another.

Finally, many systems are highly event-driven. As such, it might be apparent from the onset of the project what types of events the system must handle. This information might prove quite useful in determining which scripts to generate.

In Substep 4.3, we determine the sequencing of operations within the system, otherwise referred to as control flow. We define control flow as the aspect of a system that describes the sequences of operations that occur in response to an event. Up to this point, we have been working under the assumption that the default ordering is sequential. This is inherent in the notation we chose for scripts, which happens to present them in a fashion that leads to a sequential interpretation. There are, however, many other orderings that may be appropriate and/or required. For example, lines in a script could be executed concurrently, repetitively, selectively, or optionally.

Our goal in handling control flow is to capture the true constraints on ordering. It has been our experience that over-constraining the order in which activities take place within a system is one of the principal causes of change requests in big systems. For example, if there are 5 activities that must take place, and the order in which they take place does not matter, then there are 5! or 120 ways to execute these activities. If we just assume sequential execution, then we have chosen 1 out of the 120 potential execution paths. Over time, it is likely that some change request will occur to support one or more of the 119 execution paths we neglected. It may be difficult to actually build a system that supports all 120 pathways, but that is a design trade-off issue. At analysis time we are interested in capturing the true constraints so that designers understand what is required and what is optional.

In order to capture the true constraints on ordering, we need to annotate the scripts we created, unless we believe that sequential ordering is an appropriate default. What is needed is a notation that can capture the different types of ordering we might desire. Several are available: Petri Nets [8], Action Diagrams [8], and Statecharts [5]. Our enhancement is to associate a diagram with each script such that the diagram describes the ordering of the activities within the script.

Past, Present and Future of OBA

The OBA methodology has been evolving for the past two years. Its original incarnation was as a series of seminars produced by ParcPlace Systems. These seminars have evolved into a 3½-day course offered by ParcPlace Systems. During the past 1½ years, approximately one thousand people have attended this course or variants taught in North America, Europe and Australia. Their numerous suggestions have helped the methodology grow into its current form.

A number of organizations have successfully applied the full OBA approach from the onset of their projects, eventually yielding results coded in both Smalltalk and C++. They typically reused off-the-shelf tools to support the analysis effort. The lack of specialized, integrated tools has deterred organizations from using OBA on large-scale efforts. This is currently changing.

A prototype set of tools is currently under development with a ParcPlace client in the manufacturing sector. This prototype is being developed in Objectworks\Smalltalk Release 4 and fully supports the OBA approach. At the time of the writing of this article (May, 1992), the prototype is being tested in a number of organizations who have been trained in OBA. Based

on the outcome of the tests, we will determine revisions to both the methodology and the corresponding tools, which can then be made more broadly available.

Summary

This article discussed a methodology for analysis that we call Object Behavior Analysis. By using this approach, the following artifacts are created:

- Scripts
- Glossary of Party (Initiator-Participant) Names
- Glossary of Participants' Services
- Glossaries of Attributes
- Glossary of State Definitions
- Object Modeling Cards and various Object Relationship Diagrams
- System and Object Life Cycle Diagrams

In this article we attempted to provide an overview of a behavioral approach to object-oriented analysis. Our emphasis was on describing how, through this approach, it is possible to start from clearly stated system goals and objectives, to work with experts and end users to capture system requirements, and to turn these potentially ambiguous requests into a statement of requirements that are expressed in terms of objects, object relationships, and system dynamics, and that can be fully justified in terms of the original goals.

Acknowledgments

Our thanks to Vicki Katzman for assisting in the development of our ideas and of this article in particular, and to Brian Alexander for his editorial assistance.

References

1. Beck, K. and Cunningham, W. A laboratory for teaching object-oriented thinking. In OOPSLA '89 Conference Proceedings, *ACM SIGPLAN Note 24*, 10 (Oct. 1989).
2. Booch, G. *Object Oriented Design with Applications.* Benjamin/Cummings Inc., Redwood City, Calif., 1991.
3. Coad, P. and Yourdon, E. *Object-Oriented Analysis.* Yourdon Press, Englewood Cliffs, N.J., 1990.
4. Gilb, T. *Principles of Software Engineering Management.* Addison-Wesley, Reading, Mass., 1988.
5. Harel, D. Statecharts: A visual formalism for complex systems. In *Science of Computer Programming.* Vol. 8, No. 3, North Holland, 1987, pp. 231–274.
6. IFPUG. *International Function Point Users Group: Function Point Counting Practices Manual.* Release 3.1, Jan. 1991.
7. Jacobson, I. *Object-Oriented Software Engineering.* Addison-Wesley, Reading, Mass., 1992.
8. Kowal, J.A. *Behavior Models: Specifying User's Expectations.* Prentice Hall, Englewood Cliffs, N.J., 1992.
9. ParcPlace Systems. *Object-Oriented Methodology Course Notes.* ParcPlace Systems, Inc., Sunnyvale, Ca., 1992.
10. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, N.J., 1991.
11. Scott, A.C., Clayton, J.E., and Gibson, E.L. *A Practical Approach to Knowledge Acquisition.* Addison-Wesley, Reading, Mass., 1991.
12. Shlaer, S., and Mellor, S.J. *Object Lifecycles: Modeling the World in States.* Yourdon Press, Englewood Cliffs, N.J., 1992.
13. Wirfs-Brock, R., Wilkerson, B. and Wiener, L. *Designing Object-Oriented Software,* Prentice-Hall, Englewood Cliffs, N.J., 1990.

CR Categories and Subject Descriptors: D.2.1 [**Software**]: Software Engineering—*requirements/specifications;* D.2.10 [**Software**]: Software Engineering—*design;* I.6.0 [**Computing Methodologies**]: Simulation and Modeling—*general;* I.6.3 [**Computing Methodologies**]: Simulation and Modeling—*applications;* K.6.3 [**Computing Milieux**]: Management of Computing and Information Systems—*software management;* K.6.4 [**Computing Milieux**]: Management of Computing and Information Systems—*system management*

General Terms: Design, Methodology

Additional Key Words and Phrases: Analysis, Modeling

About the Authors:

KENNETH S. RUBIN is manager of professional services for ParcPlace Systems. Current research interests include successful software development in large organizations, by developing and communicating object-oriented methodologies related to the analysis, design and project management.

ADELE GOLDBERG is Chairman and Founder of ParcPlace Systems, providing the technology direction for the company and therefore focusing on understanding and delivering both the form and substance of object-oriented technology.

Authors' Present Address: ParcPlace Systems, 999 East Arques Ave., Sunnyvale, CA 94086-4593; email: {krubin, adele}@parcplace.com