

Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis.

Roger P. Knott Ph.D.,
Computer Science Dept,
Loughborough University,
U.K.
r.p.knott@lboro.ac.uk

Vojtěch Merunka Ph.D.,
Dept. of Information
Engineering, Czech University
of Agriculture,
merunka@pef.czu.cz

Jiri Polak,
Deloitte & Touche Czech
Republic.
jiri.polak@deloitte.cz

Abstract.

BORM Object Behavior Analysis (BOBA) is a first stage in a process of object modeling which has proved successful in a wide number of applications. This paper discusses this method and describes a simple application. One advantage of BOBA is the small number of concepts required combined with considerable expressiveness. In this way, BOBA is in the tradition established over the past few years by E-R modeling.

1. Introduction

Development of the BORM methodology started in 1993. At that time, there were in existence several 'first generation' object or semi-object-oriented analysis methods (OMT, Martin-Odell, Booch, Coad-Yourdon, Jacobson, etc). These methods were, and still are, very useful for the development of hybrid software systems. For example an object/oriented client talking to a number of relational servers. However the authors felt that these methodologies possessed two fundamental weaknesses which made them inappropriate for their own development requirements.

Firstly these existing methods did not offer sufficient support for development using a pure object-oriented language like Smalltalk. When developing systems in Smalltalk the authors often used constructs of the language like polymorphism between objects without any inheritance or object dependency. These constructs were not supported and could not be expressed in any of these existing development methodologies. Additionally, in the diagrammatic notations, they provided it was impossible to represent most pure object-oriented algorithm. Such algorithms may often be described as mutual asynchronous communications (message passing) between objects, which as the result of receiving

messages invoke internal methods with a consequential change in their state.

Secondly, these existing methodologies initially commenced with the construction of a set of classes showing inheritance and aggregation hierarchies. While this is an effective way of expressing the structure required for subsequent coding in an object-oriented language, it is not however effective in illustrating the problem domain. This is because the 'object oriented nature' of these diagrams are difficult for domain experts, not educated in computer science concepts, to understand. Consequently such diagrams cannot be used in describing proposed solutions to clients.

The initial work on BORM was carried out under the support of the Czech Academic link program of the British Council, as part of the VAPPIENS research project; further development has been carried out with the support of Deloitte & Touche Czech Republic. (VAPPIENS was funded by the British Governments CZALP academic link programme, administered by the British Council. The authors acknowledge the support they received from this source, which enabled them to meet and carry out the initial work, out of which BORM grew.)

BORM has been used for a number of large projects including

- the identification of business processes in Prague city hospitals,
- the modeling of properties necessary for the general agricultural commodities wholesale sector in the Czech Republic,
- as a tool for business process reengineering in the electricity supply industry
- as a tool for business process reengineering for telecommunication network management in the Czech Republic.

2. Developing Object Oriented Systems

Developing software systems is a complex activity fraught with many difficulties for software engineers as they endeavor to ensure that the 'right system' is built. A 'right system' being one that meets the user's needs at a cost they can afford.

On the surface this would appear a straightforward task, first year university students studying system design, are often surprised when it is pointed out to them that incorrectly specifying the required system is one of the major causes of software systems failure.

Such students, however, have little experience of the complexity of the real world where software developers and experts from the user domain appear to live in different universes, each with their own jargon, which acts as a barrier to true communication.

It is in this context that software developers face the first and perhaps major challenges of software development; to fully understand the user domain and moreover, to convey their understanding of that domain to the user.

Adele Goldberg [1] uses the term 'concept space' to describe what the user/experts believe, assumes or knows to be the case. The 'articulation space' is what the expert/user communicates in response to the analyst's questions. The analyst then constructs a model to feed back to the user/expert their mental model of the concept space, which they construct out of the information presented in the articulation space. The difference between this analyst's model and the user space is the concept gap.

To a certain extent, part of this gap is unbridgeable; we cannot easily reduce the gap between concept and articulation space as these exist in the user/expert's head. It is true, however, that the languages, natural and graphical, used by the analyst in representing this model are a vital component in the user/expert's ability to validate this model against the users own concept space.

The problem is to find a common language for the developers to express their understanding of the problem space that is both sufficiently rich for the developers to fully articulate their ideas, while also being comprehensible to users from all areas of discourse.

Use-Case[2] has become a well-accepted part of Object Oriented analysis and in many cases has proved a useful mechanism for communication between developers and domain experts. We do not intend to discuss it further here. However, Fowler[3] highlights some deficiencies in the Use-Case approach, suggesting that "*activity diagrams can be useful in cases in which workflow processes are an important part of the users' world.*"

Activities are a key component of business process modeling [4], [5]. Eeeles and Sims[6] define a business process as consisting of a number of elements; activities,

transitions, states and decisions. They too state that the UML activity-diagrams can be a useful modeling tool in capturing business processes.

2.1 The BORM Approach

BORM, like other OOA&D methodologies is based on the spiral model for the development life cycle [9]. One loop of the object-oriented spiral model contains stages of **strategic analysis, initial analysis, advance analysis, initial design, advanced design, implementation and testing.**

1. The first three stages are collectively referred to as the **expansion stages**. Expansion ends with the finalizing of the detailed analysis conceptual model, which fully describes the solution to the problem from the requirements point of view.
2. The remaining stages are called **consolidation stages**. These are concerned with the process of developing from 'expanded ideas' to a working application. During these stages, the conceptual model is step by step, transformed into a software design.

BORM was initially developed as an object-oriented method for the analysis and design of object-oriented software systems. The process starts from an informal problem specification and provides both methods and techniques, to enable this informal specification to be transformed into an initial set of interacting objects [7]. The tools and techniques developed for requirement analysis and used in the initial phases of BORM, provide an independent method for business process modeling as part of business process reengineering. The authors find that this independent method, referred to as BOBA (BORM Object Behavior Analysis) is frequently used alone.

One advantage of this latter approach is that it provides a close interactive interchange between the developers and members of the user's organization. As well as identifying initial objects, BOBA elicits from the domain experts detailed descriptions of their requirements which are fed back to them via easily understood descriptions of the proposed system's behavior, using a number of tables and graphs.

The problem specifications, from which the process starts, are obtained from relevant parties in the problem domain by interviewing. This determines a list of required system functions, which are essentially Use Cases.

From this list, a set of system scenarios is formed. BOBA scripts always include at least the four sections shown in Table 1.

Table 1

1	Initiator	A brief verbal description of the beginning of the scenario including any inputs or entry conditions. It also describes the first event or first activity of some element within the process.
2	Action	A verbal description of the process itself.
3	Participants	The set of those elements of the system, which are required for the action. It is often the case that the same participants may be present in several processes of the modeled system.
4	Result	A brief verbal description of the outcome and outputs of the scenario.

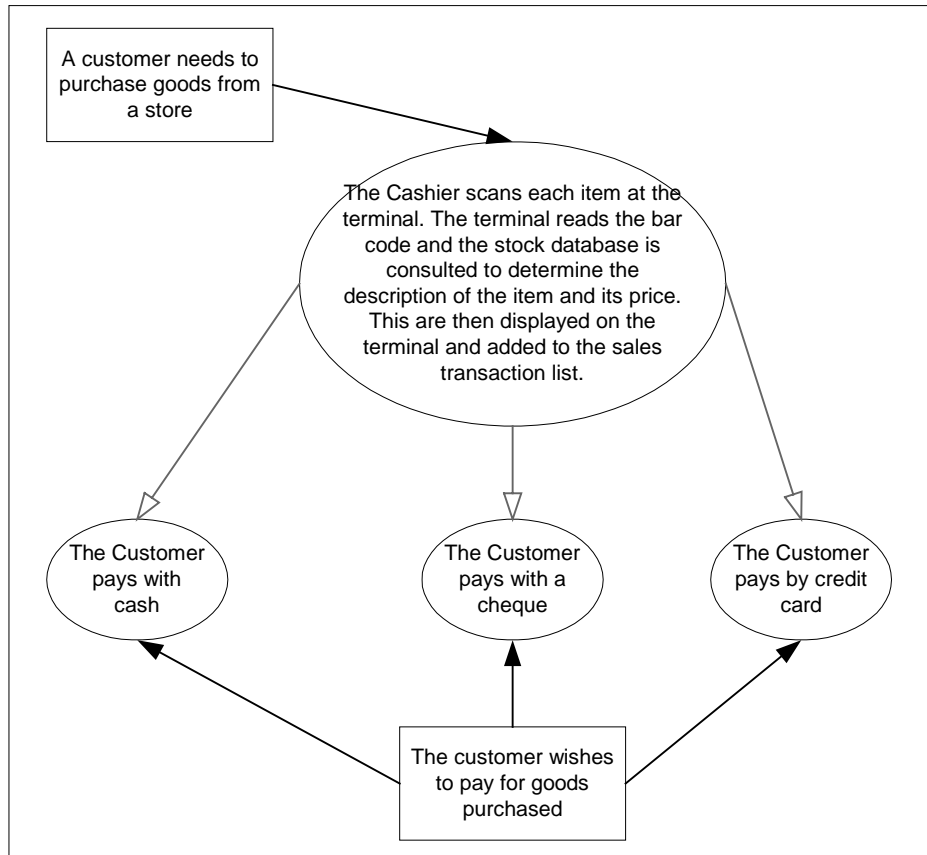


Figure 1 System Functions and Scenarios

These columns represent the four most important attributes of each scenario. The complete set of scenarios is capable of describing system behaviors, as well as determining the objects that perform these behaviors. In addition to those four attributes each scenario must also refer to the required system function it realizes.

Currently, the stages of BOBA can be carried out with the aid of the METAEDIT[®] case tool developed by Metacase Ltd (Metacase Ltd., Juväskyllä Finland, info@metacase.com, <http://www.metacase.com>)

3 A Case Study

We illustrate the details of the BOBA method through a case study. The system is the familiar one of a EPOS (Electronic Point of Sales) terminal in a supermarket.

The main functional requirement of the system to support such terminals is to process the goods that a supermarket customer brings to the terminal and to produce the total cost of this transaction.

Figure 1 shows two required system functions, together with four associated scenarios. We can click on each scenario to define and expand its description. If we

look at the description of the first scenario we have the window show in figure 2:

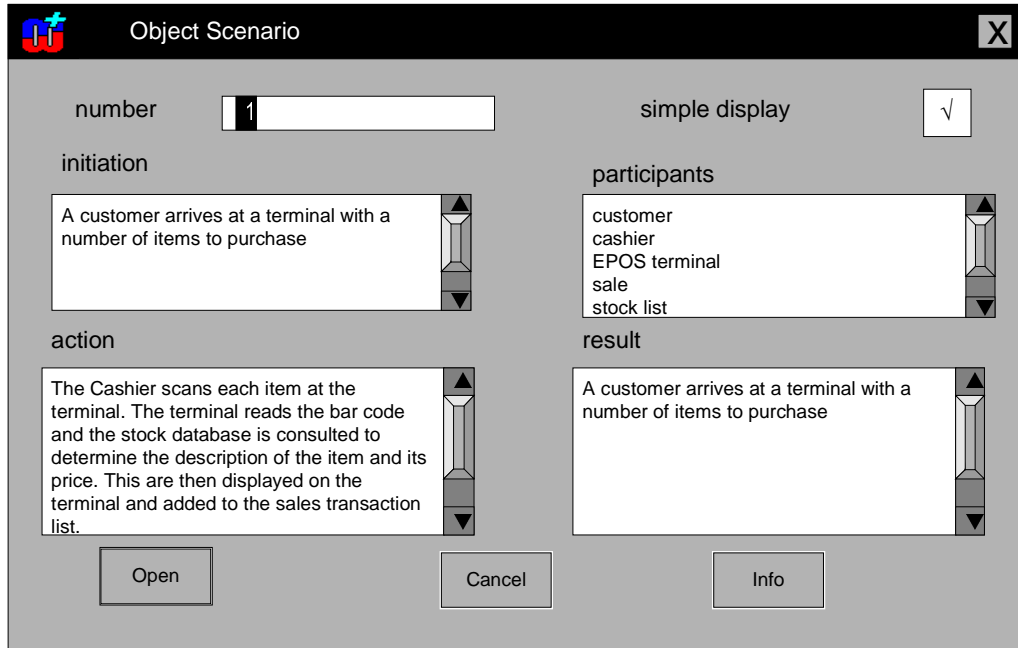


Figure 2 Properties of a Scenario

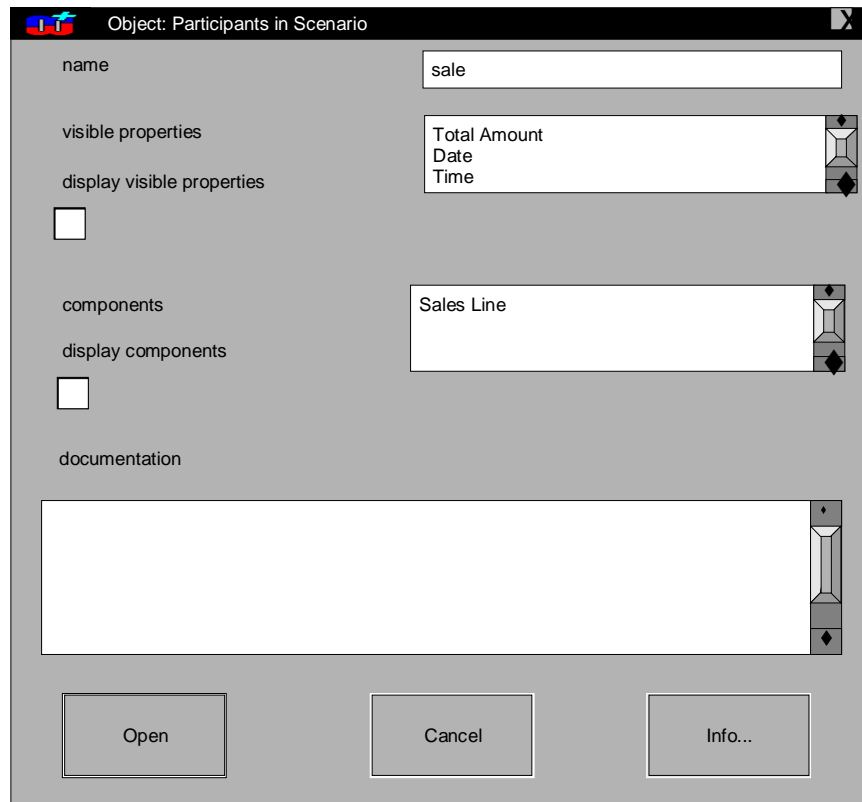


Figure 3. Object Details

Note that each of the columns previously identified for a scenario description corresponds to a pane of this window.

This table includes a list of participants. This is the term we use for what will probably become the objects in the system. This form can be completed with the user's participation and builds up an initial view of many aspects of the required system.

In BOBA, a *participant* is any entity that has a role in a scenario. These may eventually correspond to either system objects or actors. At this stage we do not distinguish. If the software is to be released in a series of incremental steps, the system boundary could be dynamic. Participants might be external to the system initially, but in later releases be implemented as part of the system.

For each participant we can add details as shown in figure 3. Here we are beginning to identify relevant attributes of the candidate objects.

3.1 Modeling Processes

One further feature of the BOBA approach, is the ability to model processes. We noted earlier that Use Case often requires the support of activity diagrams to fully convey the deeper meaning of any system. BOBA uses its own form of Business Process Modeling to achieve such deep understanding.

In BOBA process diagrams we display activities, states, transitions and decisions. A process diagram is a visual representation of object associations and communications in a particular process. The notation is very simple and easily understood by domain experts and it does not contain representations of unnecessary object oriented concepts which will only be required at subsequent stages in the development process.

In carrying out the projects described earlier, we found that to teach the notation and syntax to everyone in the consultation team, required at most half a day. This synergy of simplicity and potency, allows domain experts to become active members of the modeling team, working with formal modeling tools and techniques.

The BOBA Object Relation Diagram (ORD) – also called as 'process diagram' uses the graphical concepts, shown in Table 2.

Additionally, the initial state and terminal states are shown using the standard state transition notation.

Table 3 gives some indication of the size of the models we have constructed for some of the projects we have undertaken using BOBA.

BOBA's process model is strictly based on the theory of finite automata. Previously published work [9] and [10] support this approach. It is our contention that the object-oriented data and computational model is very similar to the concept of finite-state automata.

We view an object-oriented system as a collection of mutually communicating objects, where each object behaves as an automaton. Outputs of some automata (objects) in this system may be connected to inputs of other automata (objects). These connections are called 'communications' between objects.

In addition, we adopted the Mealy-type of automaton for modeling object behavior. This means an object is active (receives inputs and generates outputs; i.e. communicates) during transitions among its states. We associated the concept of object method with the concept of object transitions into a single concept of object 'activity'. Consequently, we put activities on transitions among states in our diagram.

It is also possible, of course, to draw object activity without any transition between states. We may choose to do this if we do not require a detailed description of some object or when cataloguing an activity that expresses a transition from a state to itself. From the formal viewpoint however, each activity performs some change, which is modeled as the transition from a state to a state, within the same object.

Thus, we have a 'two-dimensional' style of essential (business) object modeling:

The first dimension is the sequences of communications between activities of the objects in the system. This is very similar to object interaction diagrams of UML.

The second dimension is shown inside each object participating in the system. This is the sequences of related states and transition of this object. This sequence may be understood as the 'subjective' interpretation of the modeled process from the viewpoint of an object participating in that process. This is very similar to state-transition diagrams of UML.

Figure 4 shows the initial process model for the customer purchasing items from the store.

Each process diagram is related to a scenario from which it has been exploded. These diagrams have to be drawn, but Metaedit does support the drawing process as all the necessary information, which has been obtained in previous stages, can be imported from the repository.

Many aspects of the model under construction, can be reported as html documents or as MS Access database format files. One such report is the set of model cards for all the candidate objects as shown in figure 5.

These are very similar to CRC cards with the exception that they do not show collaborating objects. Collaborations are however provided in one of the other reports as shown in figure 6 below.

From these tables, we can easily construct the initial conceptual model for the proposed system using the list of objects and associations. In future versions of the case tool this will be constructed automatically.

Table 2

Concept	Symbol	Description
Object	A rectangle with name in top left corner	Objects are process participants determined from the collection of modeling cards.
State	A rectangle with state description in italics	Express the changes in objects during their life history, The state of an object is determined by the value of its visible properties and its current activity. States are drawn within the object rectangle.
Activity	An Oval	The dynamic aspect of an object required to satisfy some required system functionality. Activities are drawn within an object or a state or external to the object but connected by a continuous thin line.
Communication	A directional arrow drawn between activities	This expresses the control flow between activities. Communication may also include some data flow in the same way as in conventional object collaboration diagrams. A small arrow (object) in the appropriate direction represents such data flows.
Transition	An Arrow with a large, open arrowhead, drawn between states and directly associated with an activity	This expresses the dynamics of an object's life history. In BOBA, transitions are viewed as arising from the object having performed some activity.
Association	A thick line with a large closed arrowhead drawn between states or objects.	This expresses a static relationship between objects. The existence or nature of this association may change as the objects involved change state. It can be viewed as a special type of visible property. Association at this stage includes both 'is-a' and 'has-a' hierarchies.

Table 3

Project	Number of system functions	Number of scenarios	Number of process diagrams	Number of objects* (participants)	Average number of states per object	Average number of activities* per object
National agrarian chamber (analysis and design of software for fruit market public information system)	4	7	7	6	4	4
Hospital complex (BPR of organization structure)	6	12	12	8	10	12
TV and radio broadcasting company (BPR and company transformation for open market)	4	9	9	14	8	8
Regional electricity distribution company (customer information system analysis)	12	19	19	23	12	12
Regional electricity distribution company (failure handling information system analysis and prototype implementation)	19	31	34	27	13	14
Regional gas distribution company (BPR of all company)	28	81	97	210	11	12
Regional gas distribution company (BPR of all company)	23	60	63	120	12	12

* exclusive objects in communications data flows

** in BPR projects, each object activity contains approx. 6-10 additional attributes like business goal applicability, required job positions, time requirement etc

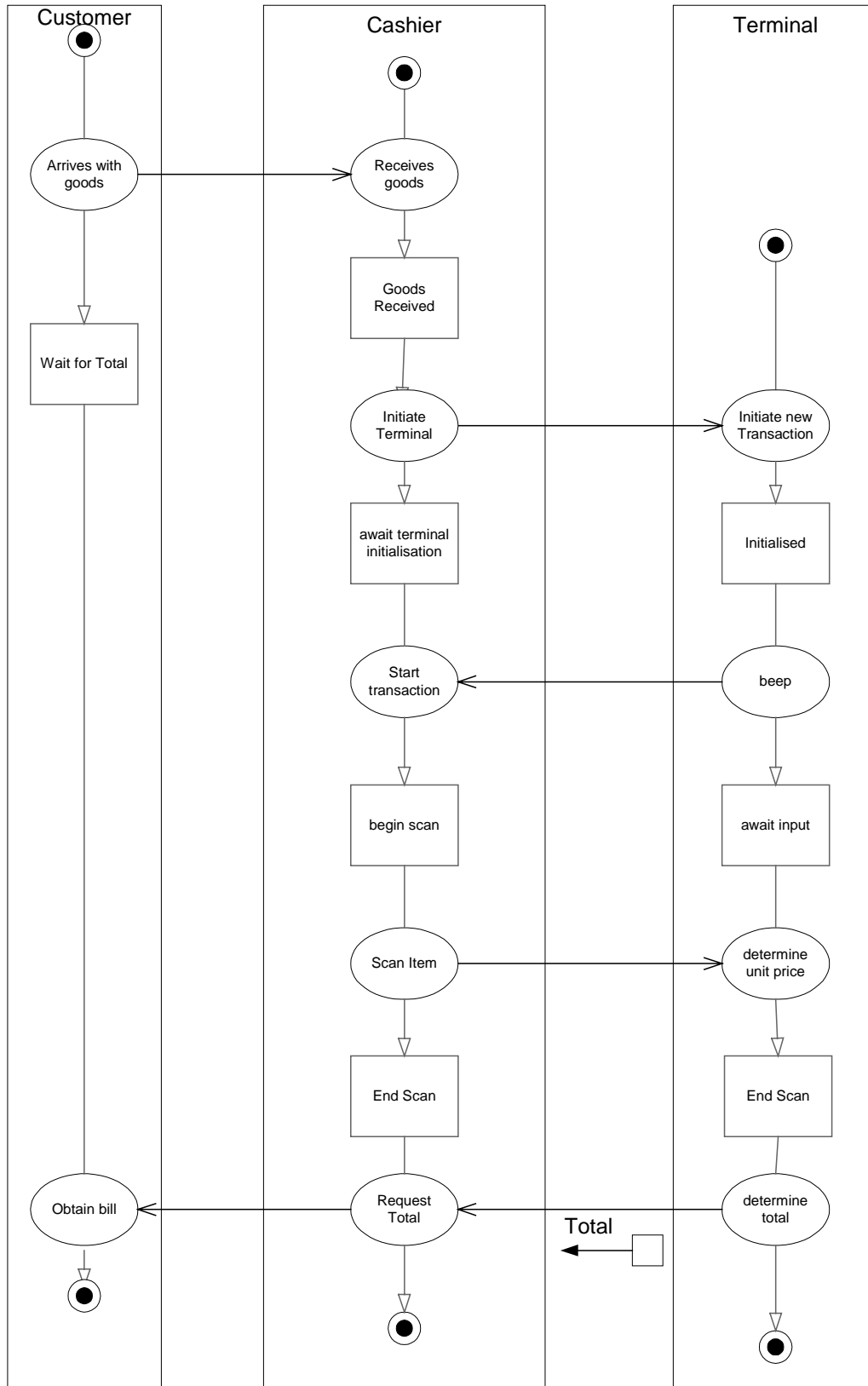


Fig. 4. The Initial Process model

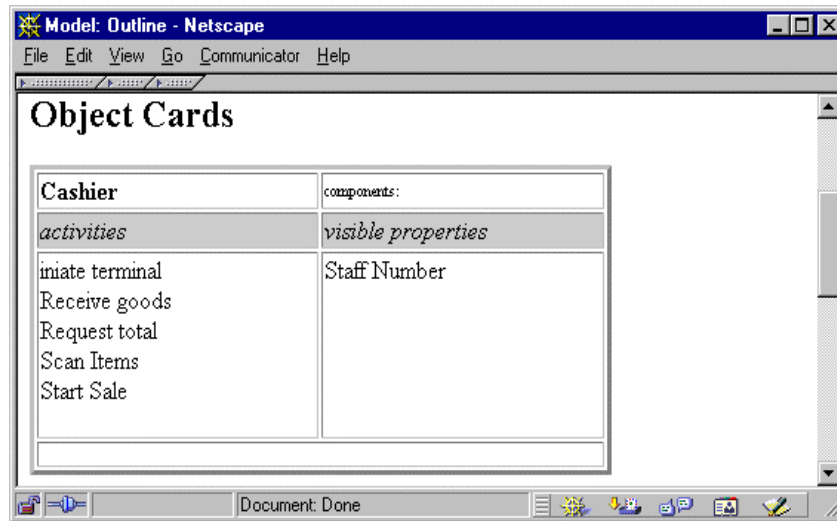


Fig. 5. Object Cards

Cashier	Cashier	Customer	EPOS Terminal
initiate terminal			
Receive goods			
Request total		x	x
Scan Items			x
Start Sale			

Figure. 6 Object Communication Cards

4 BOBA and UML

Note that the BOBA process diagrams incorporate several diagrams from the UML. If we contract the object rectangles until they become vertical lines, we obtain a UML sequence diagram showing the flow of messages between objects.

Likewise if we contract the states or the activities we can obtain activity or state diagrams respectively. By having all these features on a single diagram however, we believe a more comprehensive picture of the system behavior is obtained.

5 The Main advantages of BOBA Process Modeling

5.1. Participant History Diagram

A Participant History Diagram shows the progress of the participant through its relevant activities and states. Such a diagram can provide information on the process from the viewpoint of each of its participants. Thus the collection of these diagrams over all participants provides a complete view of the process, from the perspective of its participants. This is referred to as the **Internal View** of the process. The internal view of a process is a description of how the process looks from the inside. For participants from the user domain, these internal views

should closely correspond with their conception of how the proposed system is expected to behave. We can thus, early in the design process, validate our design against user expectation.

5.2. Process-Participant Interaction Model

We can diagram the history of all participants in a process, together with all interactions between those histories. Each interaction is a communication between activities in the histories of collaborating participants. The process itself is expressed by a sequence of such communications, flowing through several participants' histories. This flow is referred to as the processes' **activity trace**.

The consideration of the activity trace of a process, is a valuable way of identifying all the activities and states of participants necessary for the process. Many of these activities, essential for the process are external to the main process flow. We can thus ensure completeness of our design at an early stage.

5.3 Self Correcting set of Activities

BOBA's representation of a process activity trace enables the developer to easily identify who is involved in each activity and their particular responsibility to that activity. Thus the modeler is constrained by the development method and can only add activities to some participant's history and which are internally consistent with activities and stages already present in the process model.

5.4 Identification of Objects

As we saw above, BOBA contains in its repository a document detailing each participant in the system. Those participants who are not actors will become objects in the proposed system. Thus we can easily identify an initial set of objects. A task which often proves to be extremely difficult to carry out in other development methodologies.

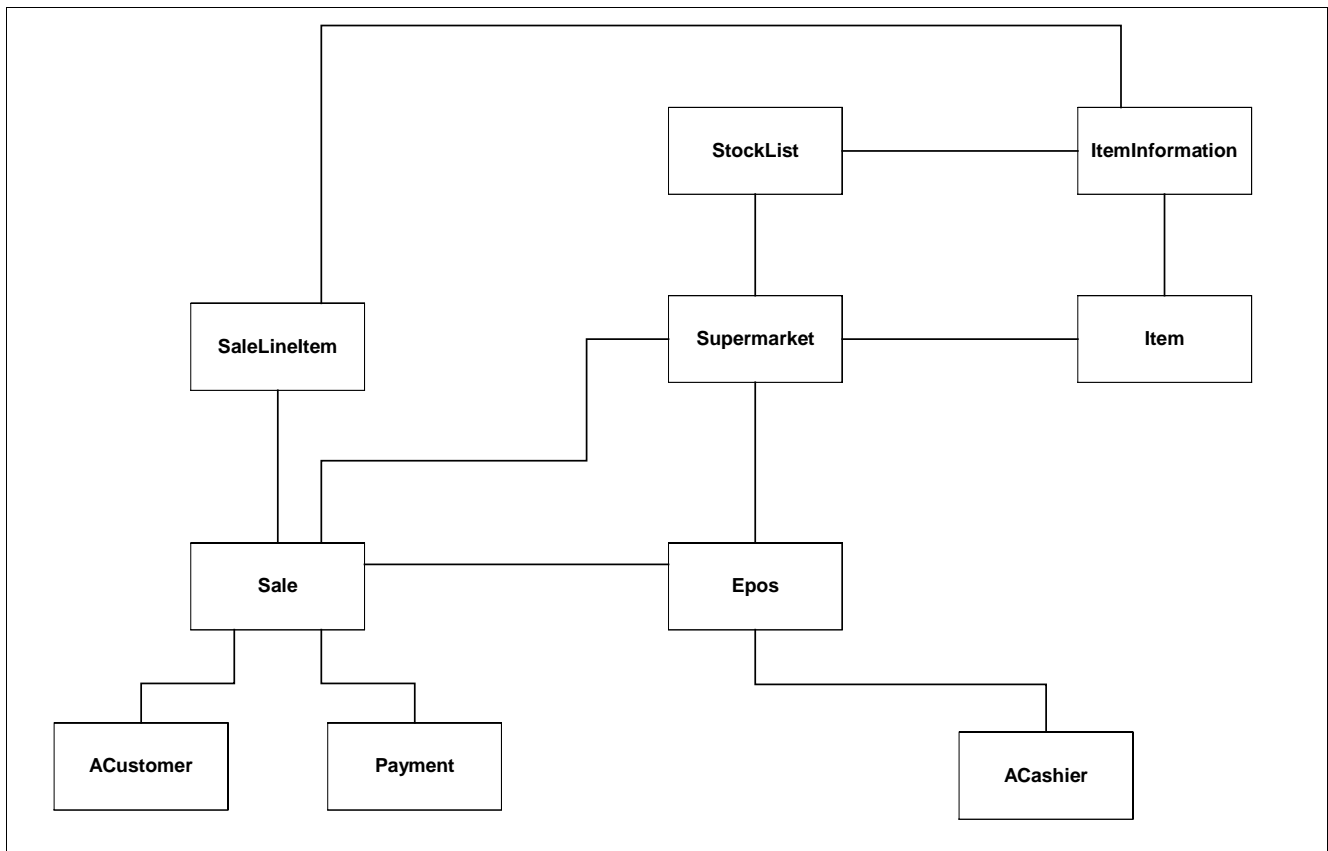


Fig 10. The Initial Conceptual Model

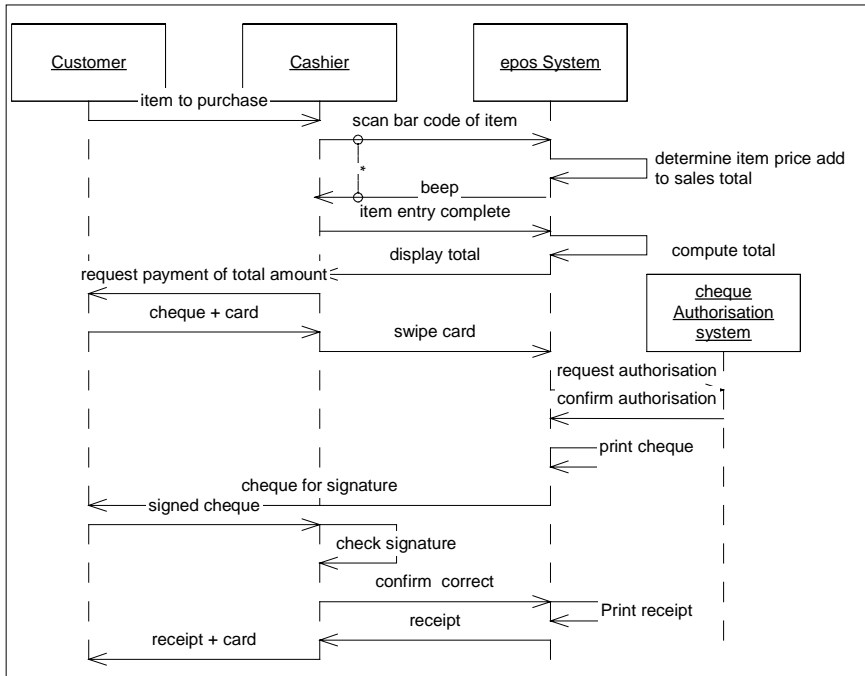


Fig. 11. A UML Sequence Diagram

6 Conclusion

Our experience since 1991, in process modeling suggests that classical conceptual diagrams are not suitable for first stages of analysis, where basic processes need to be recognized. Such diagrams are too complex for the users from the problems domain community as they often contain excessive detail concerning potential software implementations (i.e. classes, inheritance, public/private methods, attributes, link classes, etc.)

We believe that the Object Oriented Programming community needs some simple yet expressive tool for process modeling; able to play an equivalent role to that played by Entity-Relation Diagrams and Data-Flows Diagram over the past decade. One of the strengths of these diagrams was that they contained only a limited set of concepts (approx 5) and were comprehensible by problem domain experts after a few of minutes of study.

That is why we developed the BOBA process diagram. Our approach is to start with a limited set of high level concepts which can subsequently be transformed into more software-oriented concepts necessary for the construction of a software oriented conceptual model.

Our experience over a number of projects, large and small from many diverse areas, suggests that the small set of concepts we selected are sufficient for the task of fostering mutual collaboration between software developers and problem domain experts.

Thus we feel that the BOBA approach in requirement analysis and knowledge representation is an advantageous pre-cursor to subsequent object-oriented modeling at the level of object classes, inheritance, ... such as provided by various methodologies and supported by UML.

7 References.

- [1] Goldberg A., Rubin K. S.: **Succeeding with Objects - Decision Frameworks for Project Management**, Addison Wesley, Reading Mass, 1995.
- [2] Jacobson, I., Christersson, M., Jonsson, P. and Overgaard, G., **Object-Oriented software Engineering- A use Case Driven Approach**, Addison Wesley, Reading Mass, 1992.
- [3] Fowler M, **UML Distilled: Applying the Standard Object Modeling Language**, Addison Wesley, Reading Mass, 1997
- [4] Taylor, D., A. **Business Engineering with Object Technology**, John Wiley 1995.
- [5] Darnton, G., Darnton, M. **Business Process Analysis**, International Thomson Publishing 1997
- [6] P. Eeles P., Sims O., **Building Business Objects**, John Wiley & Sons, Inc., New York, 1998.
- [7] Satzinger J. W. and Orvik T. U. **The Object-Oriented Approach - Concepts, Modeling and System Development**, Boyd&Fraser 1996.
- [8] Bellin, D.; Simone, S.S.: **The CRC Card Book** Addison-Wesley, Reading Mass, 1997
- [9] Boehm, B. W.: **Software Engineering Economics**, Prentice-Hall, Englewood Cliffs, NJ 1981