David Gillibrand

# Essential Business Object Design

In most business system developments, the information model is developed separately from the functional model. Information models or data models can only capture limited business semantics (see Figure 1), which include four entities: *Customer*, *Account*, *Account type*, and *Transaction* together with the cardinality of the relationship between them, such as "A customer may have many accounts" and "An account must belong to a customer." The model implies existence dependency for both the Account and Transaction entities. However, there is very little scope to specify any behavior in the model. It would be impossible to represent a business rule that states a customer must give three-months notice before withdrawing his or her funds. A model is required that accurately describes real-world events and the objects on which they act.

**Information Architecture**

An Information Architecture [3] is one that shows the interactions between events and objects in the real world. Events happen in the real world; they are suffered or performed by real-world objects and are fairly instantaneous. They are represented in software by object methods corresponding to the real-world event. A way of representing such real-world objects and the events acting on them is by using Jackson-structured diagrams [2]. The object structure is drawn with the name of the object class in the node of the structure and the events/methods in the leaves of the structure. This shows the time-constrained way in which the events can act on an object. For example, in an Account class with the events open, close, deposit, calculate interest, and withdraw, a simple and obvious business rule as shown by the structure is that the account must be opened before money can be withdrawn or deposited. Other business rules are implicit from considering the time-ordered constraints of the events shown in Figure 2. These rules reflect the order in which events must be carried out in the real world. For an instance of the Account class the open event occurs first, followed by

**Figure 1. Banking scenario, E-R model.**

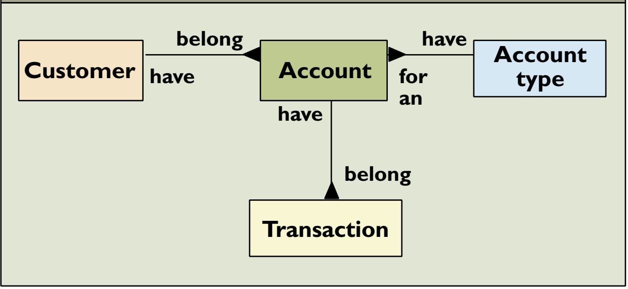


**Figure 2. Time-ordered constraints of methods within the Account class.**

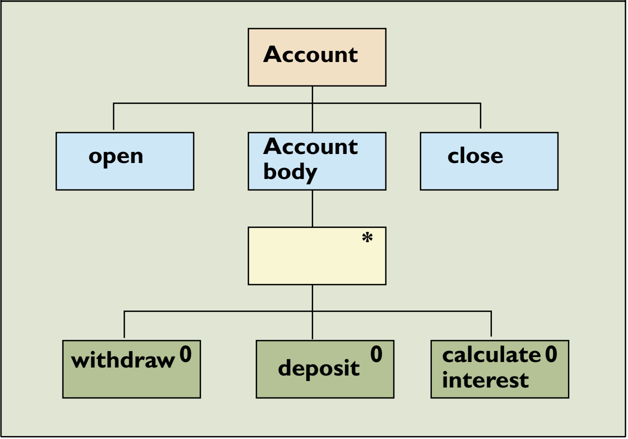
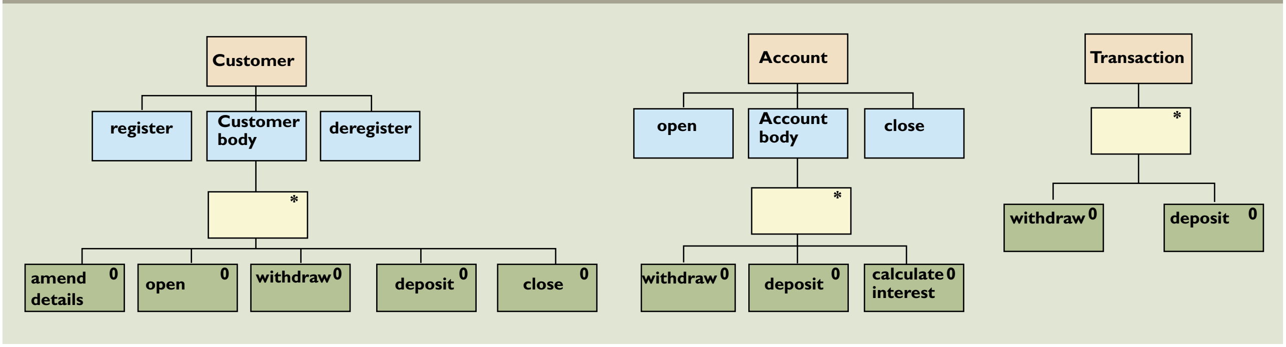


Figure 3. JSD view of the banking scenario.



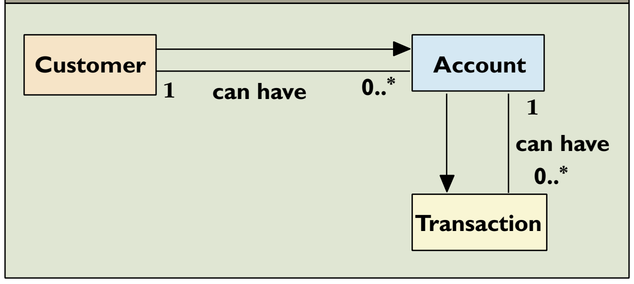
an iteration of either the withdraw, deposit or calculate interest event before the account is finally closed.

Other business rules will be implemented as pre- and post-conditions to the events taking place. For example, the open event might have the precondition that there is a minimum amount of money required to open an account. The withdraw event might have the precondition of a minimum time of notice before a withdrawal is made and the deposit event might have the precondition that there is a maximum amount of money that can be deposited per year. Post-conditions are the obligations that have to be met once the method is executed, such as the operations which update the state of the object. A deposit event will result in the increase of a data attribute "balance."

Communication between object classes is achieved by the notion of the common event. This is the simultaneous execution of the same event in all of the classes that contain the event. By way of an example consider the following simple banking scenario shown in Figure 3.

Figure 3 shows the time-ordered constraints for the events of the classes Account, Customer, and Transaction. The events for a class constitute the behavior of that particular class. The Customer and Account classes share the common events open, deposit, withdraw, and close. Account and Transaction have the common events withdraw and deposit. Conceptually the common events execute at the same time in all of the structures containing them. The effect is that the time-ordered constraints for a particular class is the sum of the time-ordered constraints imposed by all the structures.

Figure 4. Banking scenario, high-level design UML notation.



Consider the withdraw event. The Transaction class shows that the withdraw event can be followed by either another withdraw event or a deposit event. The Account class shows that an Account must be opened before a withdraw event occurs and the Customer class shows that a customer must be registered before a withdrawal is made. All these business rules are captured in Figure 3.

Note: When modeling this scenario using Jackson System Development (JSD) [2], only the Customer class and Account class would be modeled. The Transaction class would not be represented because the time-ordered constraints of its events (such as an iteration of the selection of the events withdraw or deposit) is already modeled in the structure of Account. It has no unique time dependency among its events.

Although this is a good representation of what happens in the real world, the OO paradigm is about sending messages between objects in a sequential way rather than about parallel execution. A transformation needs to take place between the logical model shown in Figure 3 and an implementation in an OO language. This can be accomplished by the common events (open, close, deposit, withdraw) only being allocated to the most appropriate class suited to their operation. This is the idea of localizing tasks to the best-suited object to handle the behavior [1]. The Customer class would be responsible for the events register, deregister, and amend details; the Account class for the events open, close, and calculate_interest; and the Transaction class for the events withdraw and deposit. The Customer class would still incorporates the cut-down methods open_account, withdraw_account, deposit_account, and close_account but they will only check that the time-ordering constraints in the Customer class are not violated (for example, a customer has to be registered before opening an account) before issuing a call to the corresponding event in the Account class.

The high-level design for the banking scenario can be represented using UML notation [4]. Figure 4 shows that the relationship between Customer and Account is one to many and that an account object is a component object of a customer object indicating an account object is a dependent object of customer. A similar relationship exists between Account and Transaction.

## Conclusion

The key to successful and maintainable business systems development relies on taking an OO approach in the analysis and design and making sure the designs produced are quality designs, that the object classes have been correctly identified and the methods allocated to them are the ones where the method behavior is most appropriate to the class. Concurrent event execution can be resolved by delegating the event to the most suitable class required to handle the behavior and just having other objects referencing the event. **C**

David Gillibrand (d.gillibrand@soc.staffs.ac.uk) is a senior lecturer in IS/databases in the School of Computing, Staffordshire University, U.K.

**References**

1. Brown G. and Forte P. Building reusable classes for frameworks. Report on object analysis and design (Nov.-Dec. 1996).
2. Jackson, M.A. *System Development.* Prentice-Hall International, Englewood Cliffs, NJ, 1983.
3. Kristen G. *Object Orientation The Kiss Method.* Addison-Wesley, 1994
4. Martin J. and Odell J.J. *Object Oriented Methods. A Foundation with UML.* Prentice Hall, 1998.

LANGUAGE TECHNOLOGY

JOINT CONFERENCE

SEATTLE, WASHINGTON

APRIL 29 – MAY 3, 2000

The Association for Computational Linguistics

www.aclweb.org

ANLP-NAACL2000

Applied Natural Language Processing and the North American Chapter of the Association for Computational Linguistics

**Refereed papers by leading scientists and engineers from around the world will be presented in two conferences:**

**ANLP – Applied Natural Language Processing**

- Monolingual text processing systems
- Multilingual text processing systems
- Spoken language systems
- Integrated NLP systems
- Tools and resources for developing NLP systems
- Evaluation of performance of NLP systems

**NAACL - North American Chapter of the ACL**

- Discourse and Dialog
- Semantics and the Lexicon
- Grammars, Parsing, and Syntax
- Corpus-Based and Statistical NLP
- Computational Psychology and HCI
- Natural Language Generation and Summarization
- Multilingual Natural Language Processing
- Spoken Language

Visit our web site www.gte.com/ANLP-NAACL2000 for details on the program, plenary speakers, Demonstrations, Exhibits, Tutorials, Workshops, and Special Millennium Events.
Contact Deborah Dahl at Unisys deborah.dahl@unisys.com for exhibits.