35. Improving Activity Diagramming with UML 2.0

Delvin Grant
DePaul University, Dgrant2@depaul.edu

35. Improving Activity Diagramming with UML 2.0

Delvin Grant
DePaul University
Dgrant2@depaul.edu

Abstract

This paper identifies four problems with UML activity diagramming that have implications for teaching and practice. Solutions to each of the four problems are provided. The proposed solutions are borrowed from existing modeling paradigms, specifically structured systems analysis and design methods (SSAD). We did not see the need to reinvent solutions when adequate remedies exist. The proposed solutions help to improve the syntax, semantics, and consistency of activity diagramming in UML.

Keywords

UML, Activity Diagrams, Object-Oriented, Activities

1. Introduction

Unified Modeling Language (UML) 2.0 is the de-facto industry standard for object-oriented systems analysis and design. The language was developed in 1997 by three researchers who combined each of their modeling languages into a new language called (UML) (Arlow et al., 2005). The youthfulness of UML has led to various problems and issues that are not well addressed. UML has been criticized for having several limitations including semantic inconsistencies, vagueness, ambiguities, and conflicting notation (Dobing et al., 2000; Price et al. 2000; Siau et al., 2001; Siau et al., 2005; Siau et al., 2006; Simons et al., 1999; Whittle, 2000). These problems can be expected of new languages since it takes time to discover and address such problems. Consequently, we decided to address some of these problems that we have encountered during our years of teaching and working in industry.

Students and analysts struggle to apply UML rules to particular problem situations. A primary inhibitor to applying the rules is the lack of understanding brought about by concepts that are vague, not explained or poorly addressed in the literature. Confusion arises out of specific modeling situations for which UML is unclear (Dobing et al., 2000; Price et al. 2000; Siau et al., 2001). This causes instructors to develop solutions to these situations with the hope of providing credible answers that make sense to students. It is common for professors to disagree on the best solution to a problem. If professors can't agree, how can we expect students and practitioners to agree? Similarly, practitioners are forced to craft solutions that are expected to make sense to team members and stakeholders. Simultaneously, the solutions should possess conceptual cleanliness which is the ability to clearly and accurately represent the problem domain (Roman, 1985). Conceptual cleanliness is a fundamental requirement of any modeling technique (Avison et al., 1986; Checkland, 1981; DeMarco, 1979). However, the limitations of UML can sometimes interfere with this fundamental requirement. There are situations where team members and students disagree with a particular solution and we know that the work of practitioners is highly influenced by what they learned in college. Therefore, if academicians cannot agree on a common solution, how can we reasonably expect practitioners and students to agree? There are times when it may be advantageous to have several solutions; however, divergent solutions can be a source for problems especially in the absence of clearly defined and generally acceptable rules. Consequently, leaving instructors, students, and practitioners to fend for themselves is a recipe for problems and project failure.

Here are two examples of problems that may arise. First, every language including UML has rules of syntax and semantics that govern the correctness and use of the language. The absence of rules makes it difficult to judge the correctness of diagrammatic solutions. When individuals disagree on a particular solution, how do they resolve their differences when rules are vague or nonexistent? Differences can be readily resolved when there are clearly defined rules of syntax and semantics. Second, it is easier to teach, understand and use a language when clearly defined rules exist. It is also easier to objectively judge the correctness of a solution; the absence of rules makes it problematic to objectively compare different solutions. These problems pose significant challenges for instructors who are trying to be fair and objective in evaluating students' work. The problem is compounded when we consider that each instructor, student, or practitioner may have divergent solutions to a problem. When these situations arise, how do we effectively resolve them and how much time and effort are wasted? I recalled a task team spending two weeks trying to figure out where a particular reference point was located on a printed circuit board. The rules of syntax governing its location were ambiguous. These are some reasons why we are motivated to find solutions to these problems by developing and refining rules of syntax and semantics to improve UML activity diagramming. To accomplish this we rely on our experience and knowledge of modeling techniques including SSAD (DeMarco, 1979; Ganes et al., 1979) and structured analysis design technique (SADT) (Ross et al., 1977).

In this paper we address four UML activity modeling problems that relate to teaching and practice. While other UML problems unrelated to activity modeling exist, they are not the focus of this research. Due to the length of this paper we decided to focus our research on four activity modeling problems that we are aware of. This does not mean that others problems, unknown to us, may not exist. The contribution of this research is to improve syntax and semantics rules of UML activity modeling. The rest of the paper is organized as follows: Section 2.0 is a literature review; Section 3.0 discusses the problems and their modeling implications for learning and using the language; and Section 4.0 provides some possible solutions to the problems. The paper ends with a discussion in Section 5.0.

2. Literature Review

Systems analysis and design is part of the systems development life cycle process that governs the development of information systems (IS) projects. Traditional systems development can be traced back to SSAD (DeMarco, 1979; Ganes et al., 1979; Ross et al., 1977; Yourdon et. al, 1978) and is still practiced today (Marakas, 2006; Whitten et al., 2008;). Pioneers like Yourdon and Constantine (1978), Checkland (1981), Bubenko (1986), Avison and Wood-Harper (1986), and Avison and Fitzgerald (1988) have made significant contributions to the systems development life cycle process. Object-oriented programming languages (OOPL) such as Simula and Smalltalk were a precursor to object-oriented systems analysis and design. The semantic gap that existed between OOPL and traditional development methods paved the way for object-oriented systems analysis and design methods. In the late eighties and early nineties, several object-oriented analysis and design methods were developed. They include object-modeling technique (OMT) for software modeling and design (Rumbaugh et al., 1991), Booch Method (Booch, 1994), and Yourdon Method (Coad and Yourdon, 1991). These methods took a different view of systems development based on the concept of an object which is the encapsulation of operations and data. In 1995 Rational Software™, now part of IBM, commissioned Rumbaugh, Booch, and Jacobson to create UML. While UML is built on the concept of the object it has been heavily influenced by traditional systems analysis and design methods.

3. Problems and Solutions

Problem 1: The Decision Problem

The decision node is a primary construct used in activity diagramming and is represented by a diamond. The diamond connects a controlling activity to subordinate decision activities. A fundamental rule of UML requires all activities to have at least one input and one output (Arlow et al., 2005; Dennis et al., 2009; Jacobson, et al., 1999; Rumbaugh et al., 2005). This rule goes back to dataflow diagramming (DeMarco, 1979; Ganes et al., 1979) and the reason for it is simple: inputs are required to perform business activities which are transforming mechanisms. Activities produce outputs which may serve as inputs to other business activities. In figure 1, the activities have no inputs or outputs so they cannot perform their processing function. Also, the controlling activity cannot communicate with its subordinates since there is no flow of information between them. Figure 1 is syntactically and semantically incorrect.

The background to this problem is that leading text books (Arlow et al., 2005; Brown, 2002; Dennis et al., 2005; Dennis et al., 2009; Jacobson, 2000; O'Docherty, 2005; Rumbaugh et al., 2005) violate this input/output rule (see figure 1). The violation of this rule has had a negative influence on students because they often leave off the inputs and outputs. From my experience, this is by far the most common problem with respect to the creation of activity diagrams by students. When UML syntax is violated it creates interpretation problems for readers, in particular the students who are learning the language. It also creates interpretation and development problems for practitioners.

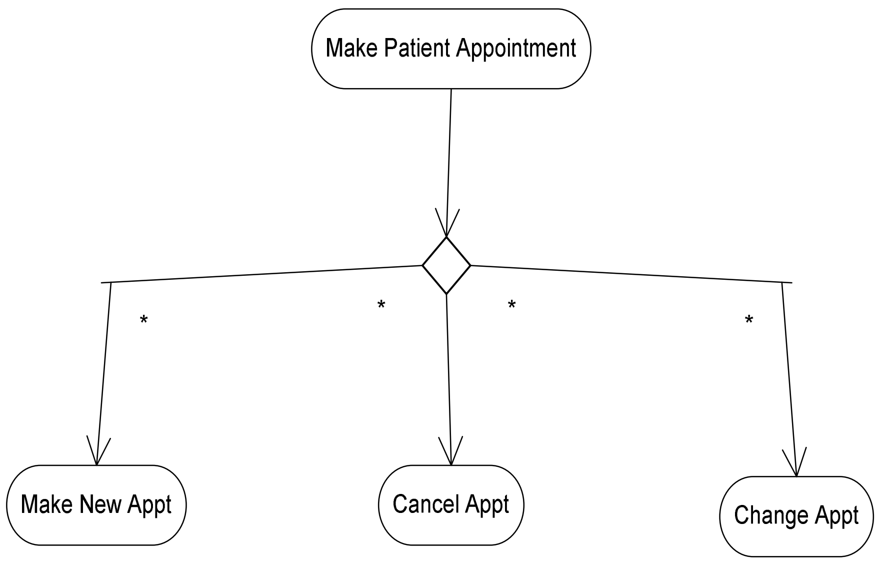


Figure 1

There are several possible solutions to the decision problem but each solution comes with additional issues or limitations. Due to the page limitations of this paper we will forgo discussion of the various solutions and present the one we consider best.

Solution 1

The recommended solution shown in figure 2 is a compromise because it is syntactically and semantically correct and reduces clutter. This is accomplished by using double-headed arrows to connect each object node to an activity, suggesting a two-way flow of information. The semantically correct use of double-headed arrows would indicate the passing of information between controlling activities and subordinates (DeMarco, 1979; Ganes et al., 1979). Syntactically this will help to enforce the rule that all activities require input and output.

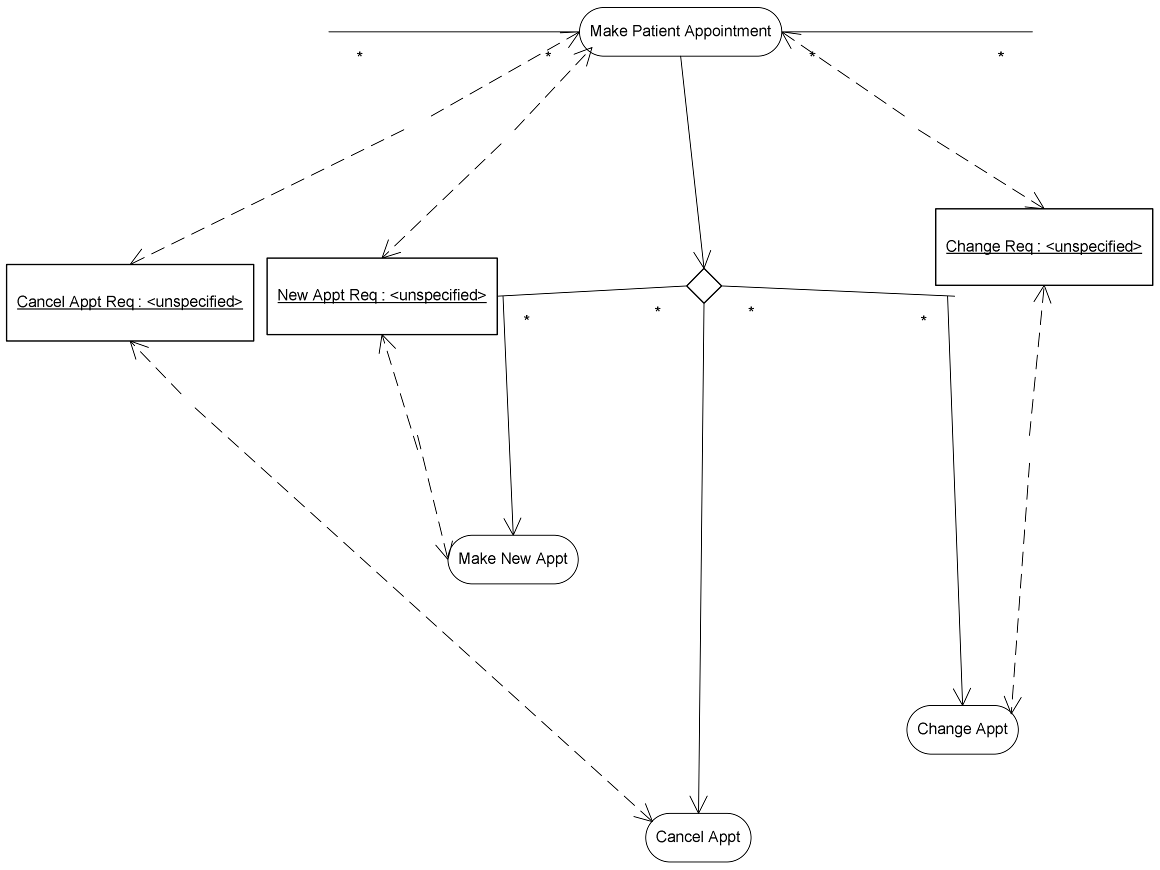

Make Patient Appointment
Cancel Appt Req : <unspecified>
New Appt Req : <unspecified>
Change Req : <unspecified>
Make New Appt
Change Appt
Cancel Appt

Figure 2

Problem 2: The Actor Problem

UML rules do not require actors to be shown on activity diagrams. Actors are shown for the first time on the use case diagram. An activity diagram is similar to a dataflow diagram in its attempt to represent business processes and activities. There are several reasons why the absence of actors from activity diagrams is poor practice. First, actors provide input to business activities and receive output from them; they serve as triggers for activating business processes such as withdrawing money from an ATM machine or placing an order. Second, knowing the association between actors and activities is important for defining the project scope, a critical success factor for projects (Clifford et al., 2006; Shtub et al., 2004). Project scope influences the choice of actors and stakeholders, the required resources, and the length of the project. A poorly defined project scope negatively impacts projects that sometime end in failure. Third, actors make it easy to identify stakeholders when questions or ambiguities about aspects of the system arise. Since business modeling requires interviewing users, it is important to identify them early in the process.

The background to this problem is that activity diagrams were not part of UML when it was first conceived because its importance was underestimated. It was not until UML 2.0 that the originators of the language recognized the importance of activity diagrams and decided to add them. Because of this historical fact UML continues to emphasize use case diagrams as the most important diagram because they served as the basis for developing other diagrams (Dennis et al. 2009). Consequently, use case diagrams are still used as the basis for defining the project scope (Dennis et al., 2009, O'Docherty, 2005) a left over idea from when the language was first created. This explains why use case diagrams have system boundaries and activity diagrams do not. This is inconsistent with other business modeling techniques such as dataflow diagrams (DFD) and activity modeling in IDEF0 (Ross et al., 1977). These techniques correctly addressed the issue of scope in the first diagram called the context diagram (DeMarco, 1979; Ganes et al., 1979; Kendall and Kendall, 2008; Yourdon et. al, 1978). Consequently, the context diagram in DFD is equivalent to the activity diagram in UML. Since use case diagrams are no longer the first diagram, the emphasis on defining scope should be moved to the activity diagram. It makes sense to define the project scope at the outset to avoid problems related to the deployment of resources and the development of diagrams that follow. Why waste resources to model business activities that are not part of the problem. In fact, it is the activity modeling process that guides the decision about which activities to include in the problem domain.

Hence it makes sense that activity diagrams should have a system boundary to represent the project scope, and the actors that interact with the system should be shown outside the boundary. In practice, a one-to-one mapping between activities and use cases exists. Activity diagrams serve as a checking mechanism for use case diagrams and vice versa (Selonen et al., 2003). For example, if activities on an activity diagram are to be computer supported they will be shown on the use case diagram. Activity diagrams have object nodes that eventually show up as classes on class diagrams. Therefore, activity diagrams are an excellent source for identifying classes (Arlow et al., 2005) that represent data to be stored by the organization. There is also a connection between the objects on the activity diagram and the objects on the scenario and communication diagram. Hence, a strong case can be made for the activity diagram becoming the foundation for UML modeling because it has more direct connections to the later diagrams than the use case. The current emphasis on use case diagrams as the foundation for the creation of UML diagrams is misplaced. It is an unfortunate consequent of the historical nature of UML development.

Solution 2

The solution shown in figure 3 is to include actors on activity diagrams to establish the connection between them and the activities they use. When activities become use cases, the connection between them would be established, thus allowing for some degree of completeness and consistency checking between the two diagrams. It makes more sense to identify actors on the activity diagram instead of waiting until the development of the use case diagram. Identifying actors is critical to defining the project scope and that should start with activity diagrams. Scope is usually defined by a system boundary that separates the problem domain or system of interest from its environment; therefore we have included a system boundary in our solution. Everything outside of the boundary is external to the problem domain or system of interest. The boundary also serves to limit the scope of the later diagrams that make up the system. To successfully model a business process it should be well bounded and clearly understood, the result of proper scoping. Successful modeling techniques like DFD's and SADT have defined the project scope on the first diagram called the context (DeMarco, 1979; Ganes et al., 1979; Ross et al., 1977). Identifying actors early in the process reduces development time and duplication of effort because the same actors show up on use case, sequence, and communication diagrams.

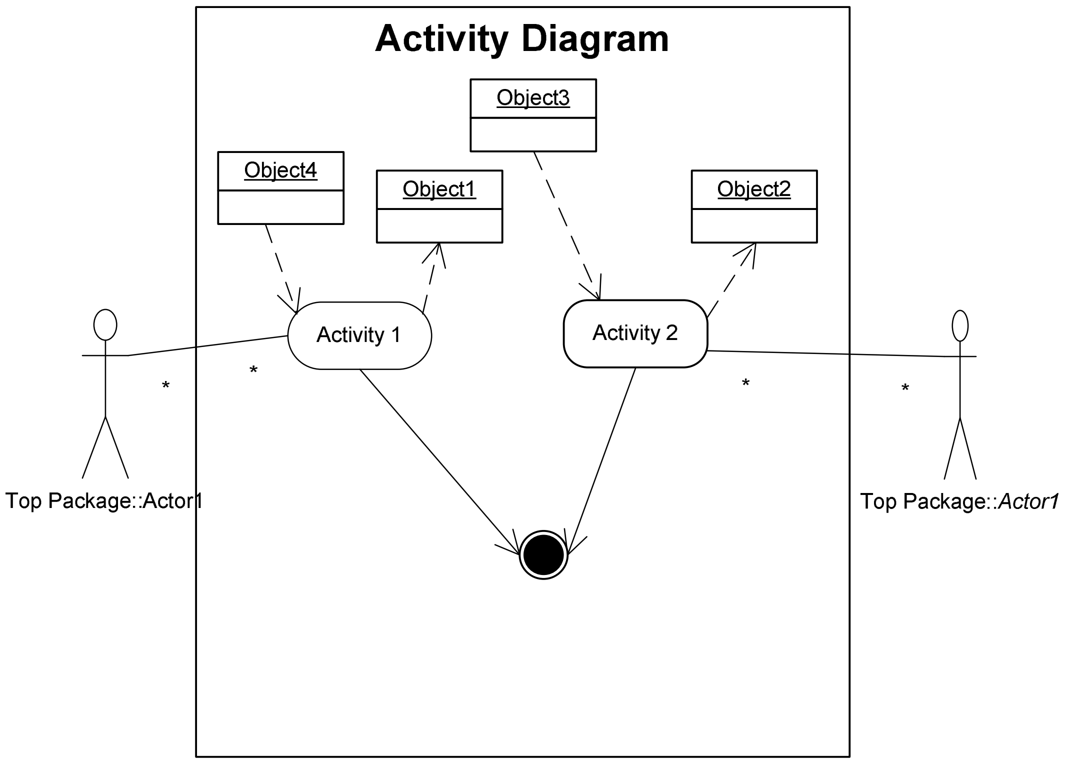


Figure 3

Problem 3: Object Flows Pointing to End Nodes

Object flows in UML are used to designate the movement of information and data between activities. The object flow symbol is a dotted line with an arrow indicating the direction of information flow. The actual information is represented by object nodes which are forms, reports, letters, memos, policy documents, best practices, emails, and files that people use to store and disseminate information for decision making. A business process is a related set of activities and the termination of the business process is represented by an end node which resembles a bulls-eye symbol (Bennett et al., 2001, Rumbaugh et al., 2005, p.158). There are examples in the literature where object flows are connected to end nodes (Dennis et al, 2005; Dennis et al., 2009, p. 161,).

The background for this problem is that such connections not only violate the rule that object flows should be connected to activities and object nodes (Rumbaugh et al., 2005), but it does not reflect reality. Information cannot be sent to end nodes because they do not exist in reality. End nodes are conceptual ideas used to designate the end of a business process. In organizations data and information are sent to people, and activities. Object flows connected to end nodes may suggest missing activities or actors that may or may not be outside the scope of the problem domain. Every business process ends with one or more activities that terminate the process. A process or activity by definition has a beginning and end (DeMarco, 1979; Rumbaugh et al., 2005). It is possible that a business process may not always end with the same activity, because different paths through a process may exist (Dennis et al., 2009). Nonetheless, a business process often ends with an activity that is connected by a control flow to an end node. In cases where several activities may terminate a business process, each activity should be connected to the end node using a control flow. Because there are so few examples of activity diagrams with object nodes it was extremely difficult to ascertain if this is a pervasive problem of UML. The incorrect and correct solutions are shown below in figure 3.0.

Solution 3
The solution shown in figure 4 is to remove the object flows that connect object 1 and 2 to the end node. Then connect activity 1 and activity 2 to the end node using control flows. This solution adheres to the rules of modeling and UML that require any activity or process to possess the necessary inputs and outputs (DeMarco, 1979; Ganes et al., 1979, Rumbaugh et al., 2005). A fundamental rule of information processing is that an activity cannot function without input and when it processes the input it produces output (DeMarco, 1979). This solution is consistent with how companies function.

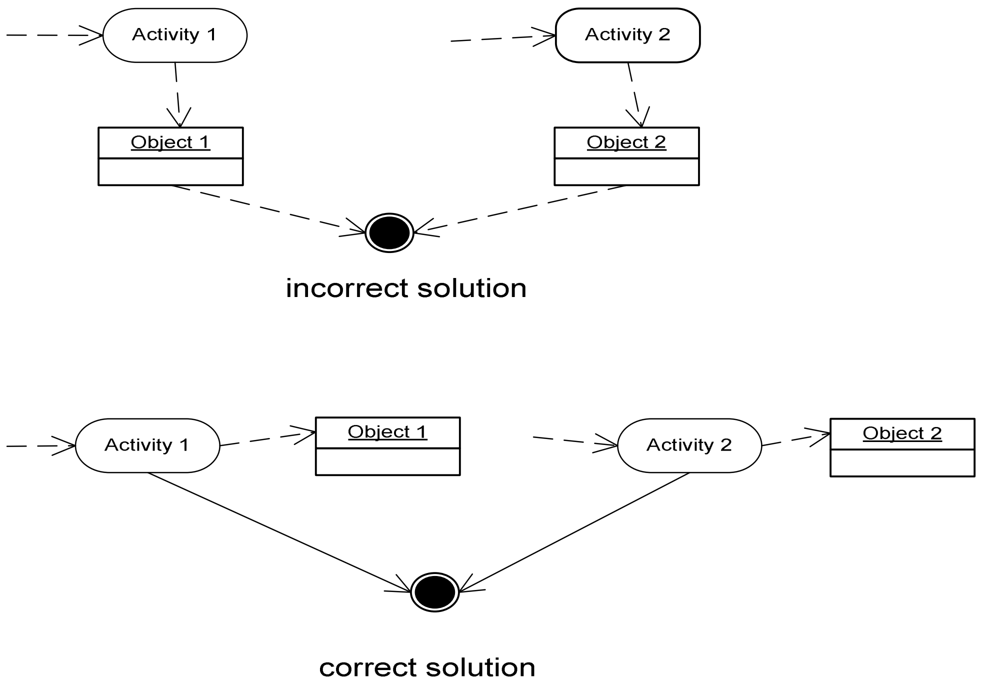


Figure 4

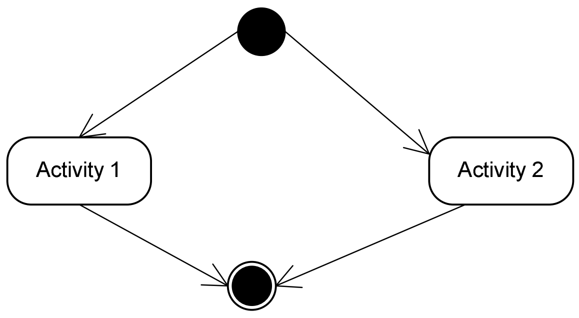
Problem 4: Lack of Activity Inputs and Outputs
Activities in UML should have inputs and outputs (Rumbaugh et. al, 2005). In UML inputs and outputs are represented by object nodes and their direction of flow is indicated by flow

nodes. This is an old rule that dates back to structured methods and systems thinking (Checkland, 1981; DeMarco, 1979; Ganes et al. 1979). The rule states that an activity or process should have at least one input and at least one output (DeMarco, 1979). The idea is that an activity must have the necessary and sufficient inputs to produce the required outputs. Dennis et al. (2008) identify two types of activities: black holes and miracle activities. Black holes accept input but never produce output while miracle activities produce output but have no input. It is common to find inputs existing in the minds of users, such as a product, discount or employee code used by a cashier at a point-of-sale register. This is why interviewing users is so valuable in the analysis and requirements process. It is surprising to find the input/output rule frequently violated in UML textbooks (Dennis et al 2008 and 2005; O'Docherty, 2005; Rumbaugh, et al., 2005; Satzinger et al., 2005). This violation is disappointing for educators who see the need to stress the importance of object nodes in the construction of activity and other UML diagrams. An object cannot exist without a class (Rumbaugh, et al., 2005) so the objects on the activity diagrams are the inspiration for identifying classes. The absence of classes impacts the construction of sequence and communication diagrams. Lastly, inputs and outputs enhance the understanding and completeness of activity diagrams, and aids consistency checking between various diagrams.

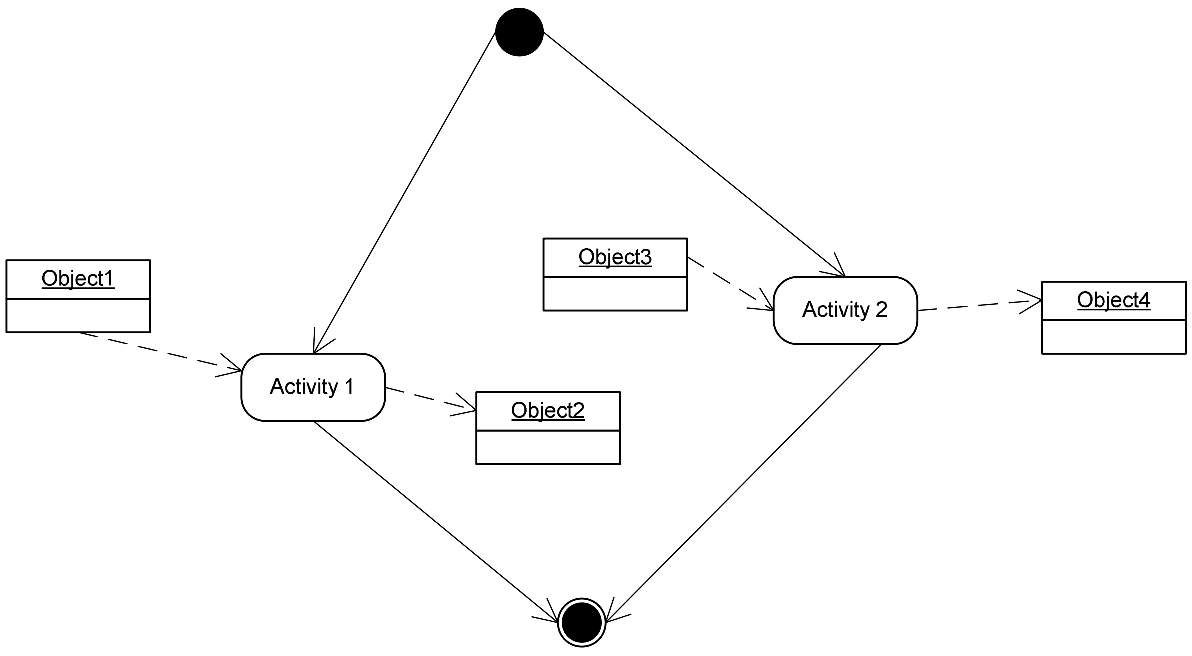
I can only surmise the reasons for this persistent violation. From my experience inputs and outputs are the most challenging aspects of developing activity diagrams. Students can readily identify activities but find it very challenging to identity inputs and outputs because they are often less visible and require more effort to identify. Identifying objects is a tedious and meticulous undertaking. In conceptual modeling inputs and outputs are expressed in logical terms and this is a challenge for many people because we are conditioned to identify objects in physical terms. It is easier to identify a physical output than a logical one. For example, many students struggle to identity the output produced by an activity that verifies a customer order. Most students would have the customer order as input and the customer order as output. This would suggest that the activity (i.e., the processing function) is non-value added. The correct output should be labeled "verified customer order," indicating that the order has been verified, a value add activity. It takes a few iterations before students get the hang of this idea. The solution, shown in figure 5, is for academicians, authors, practitioners, and students to make a concerted effort to include all inputs and outputs, thus enforcing the rules of UML. This will go a long way to better educate our students while enhancing the quality of UML diagrams that are syntactically and semantically correct.

4. Conclusions/Discussion

In the paper we identified four problems related to UML activity modeling. The problems relate to areas of activity modeling that are vague. Each problem has implications for teaching and practice. Teaching and practice require formal rules that guide professors, practitioners, and students to a common acceptable solution. In the absence of rules, solutions are arbitrary and subject to multiple interpretations. Graphical languages like UML were invented to overcome multiple interpretations often associated with spoken languages. In practice, formal rules improve interpretation and increase the likelihood that practitioners can agree on a common solution. Practitioners and students struggle with how best to deal with these issues and their solutions leave much to be desired as they vary within and across companies; hence this is another reason for rules and standards that deal with specific issues. We identified each problem and its implications for teaching and practice and provided solutions. Most solutions were borrowed from SSAD methods. We did not see the need to invent new rules and constructs when existing and familiar ones exist. The proposed solutions should improve the semantics and syntax of activity diagramming and UML in general.



Incorrect slution



Correct solution

We believe the suggested solutions would work well in teaching and practice. We have used them to teach our students and they have benefited tremendously. The first benefit is that students get into the habit of applying the rules consistently because when they don't we penalize them. Consequently, I have seen constant improvement in the quality of activity diagrams. It is difficult to disagree that diagrams with object nodes are more complete and informative. A second benefit is that solutions which subscribe to the rules of UML enhances the quality of diagrams, the body of knowledge on UML, and improves industry practice. Lastly, we have removed the ambiguity of applying UML activity diagramming rules that are vague or have been ignored in the literature. Future research could investigate if the proposed solutions produce better quality diagrams among students than the status quo. We could also investigate if the suggested improvements work well in industry, if they are of better quality and more informative than diagrams that ignore our solutions.

References

Arlow, J, and Neustadt, I., (2005) UML *2 and the Unified Process: Practical Object-Oriented Analysis and Design*, *2nd edition*, Addison-Wesley, NJ

Avison, D.E., and Wood-Harper, A.T., (1986) *Multiview – An Exploration in ISD*, Australian Computer Journal, (18)4

Avison, D.E., Fitzgerald, G., and Wood-Harper, (1988) Information Systems development: a toolkit is not enough, Computer Journal, (31)4

Bennett, S., Skelton, J., and Lunn, K., (2001) *Schaum's Outline of UML*, McGraw-Hill, NY, USA

Booch, G., (1994) *Object-oriented Analysis and Design with Applications*, *2nd edition*, Benjamin & Cummings, Redwood City

Brown, D., (2002) *An Introduction to Object-Oriented Analysis: Objects and UML in Plain English*, Wiley, NY, USA

Checkland, P., (1981) *Systems Thinking, Systems Practice*, John Wiley, Chichester, NY

Clifford, F., and Larson, E., (2006) *Project Management: The Managerial Process*, Irwin McGraw-Hill, *3rd edition*, NY, USA

Coad, P., and Yourdon, E., (1991) *Object-Oriented Analysis*, *2nd edition*, Yourdon Press, Englewood Cliffs, NJ

Dennis, A., Wixom, B., and Tegarden, D., (2005) *Systems Analysis and Design with UML version 2.0: An Object oriented Approach*, *2nd edition*, Wiley, NY

Dennis, A., Wixom, B., and Tegarden, D., (2009) *Systems Analysis and Design with UML version 2.0: An Object oriented Approach*, *3rd edition*, Wiley, NY

DeMarco, T. (1979) *Structured Analysis and Systems Specification*, Prentice Hall, NJ

Dobing, B., and Parsons, J., (2000) Understanding the Role of Use Cases in UML: A Review and research Agenda, *Journal of Database Management*, (11)4, pp. 28-36

Ganes, C. and Sarson, T., (1979) *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, NJ

Jacobson, I., (2000) *The Road to the Unified Software Development Process*, Press Syndicate of the University of Cambridge, Cambridge, UK

Jacobson, I., Booch, G., and Rumbaugh, J., (1999) *The Unified Software Development Process*, Addison-Wesley, Boston, USA

Kendall, K., and Kendall, J., (2008) *Systems Analysis and Design*, *7th edition*, Prentice Hall, NJ, USA

Marakas, G.M., (2006) *Systems Analysis and Design: An Active Approach*, *2nd edition*, McGraw Hill, NY, USA

O'Docherty, M., (2005) *Object-Oriented Systems Analysis and Design: Understanding Systems Development with UML 2.0.*, John Wiley, West Sussex, England

Price, R., Tryfona, N., and Jensen, C., (2000) Extended Spatiotemporal UML: Motivations, Requirements, and Constructs, *Journal of Database Management*, (11)4, pp.14-27

Roman, G., (April, 1985) A Taxonomy of Current Issues in Requirements Engineering, *IEEE Computer Society,* pp. 14-21

Ross, D., and Schoman, K., Structured Analysis for Requirements Definition, *IEEE Transactions on Software Engineering*, (3)1, pp. 6-15

Rumbaugh, J., Jacobson, I., and Booch, G., (2005) *The Unified Modeling Language Reference Manual*, *2nd edition*, Addison Wesley, NY

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, (1991) W., *Object-Oriented Modeling and Design*, Prentice Hall, NJ

Selonen, Petri, Koskimies, K., Sakkinen, M., (July-Sept, 2005) Transformations Between UML Diagrams, *Journal of Database Management*, (14)3, pp. 37-55

Shtub, A, Bard, J., and Globerson, S., (2004) *Project Management: Engineering, Technology and Implementation*, *2nd edition*, Prentice Hall, NJ, USA

Siau K., and Cao, Q., (2001) UML – A Complexity Analysis, *Journal of Database Management*, (12)1, pp. 26-34

Siau, Keng, Erickson, J, and Lee, L.Y., (July-Sept, 2005) Theoretical vs. Practical Complexity: The Case of UML, *Journal of Database Management*, (16)3, pp. 40-57

Siau, K., and Loo, P., (summer, 2006) Identifying Difficulties in Learning UML, *Information Systems Management*, pp. 43-51

Simons, A, and Graham, I., (1999) *Things that go wrong in Object Modeling with UML 1.3*, Chapter 16. In: Kilov H. Rumpe B and Simmonds, *1st edition*, Precise Behavioral Specification of Business and Systems, Kluwer Academic Publishers

Simons, A., (1999) *Use cases considered harmful, Proceedings of the 29th conf tech, Object-oriented Programming language and System*, Eds. R Mitchell, A C Wills, J Bosch and B. Meyer, Los Alamitos, CA: IEEE Computer Society

Whitten, J., and Bentley, L., (2008) *Introduction to Systems Analysis and Design*, *1st edition*, McGraw Hill, NY

Whittle, J., (2000) Formal Approaches to Systems Analysis Using UML: An Overview, *Journal of Database Management*, (11)4, pp. 4-13

Yourdon, E., and Constantine, L.L., (1978) *Structured Design: Fundamentals of a Discipline of Computer program and Systems Design, 2nd edition*, Yourdon Press, NY