Introduction to Design And Activity Diagrams

Davide Falessi

Overview

- *Requirements Analysis* determines what end users want and need from a software system.
- *Specification* formally defines user requirements.
- *Design* defines and organizes the major operational components of the system.
- *Implementation* defines operational details in a programming language.

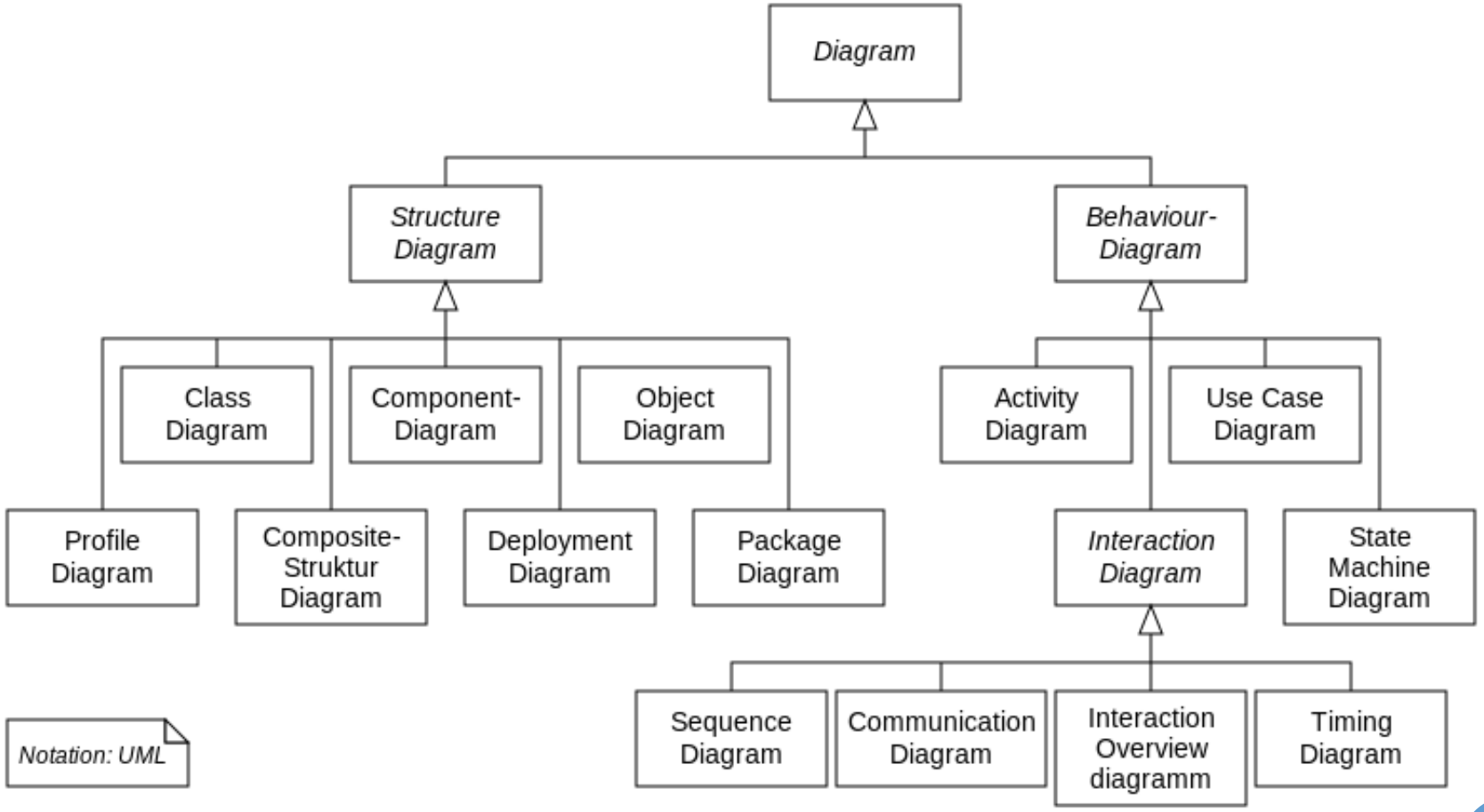
Major goals of the design process

1. **Traceability**: elements of the design trace back to corresponding elements of the specification.
2. **Modularity:** elements of the design are organized into logically cohesive modules.
3. **Portability**: the design is sufficiently general that it can be implemented on a variety of platforms and a variety of (related) programming languages.
4. **Maintainability**: the system is designed such that it can be easily repaired and enhanced.
5. **Reusability**: where appropriate, modules are designed to promote their reuse in other (future) designs.

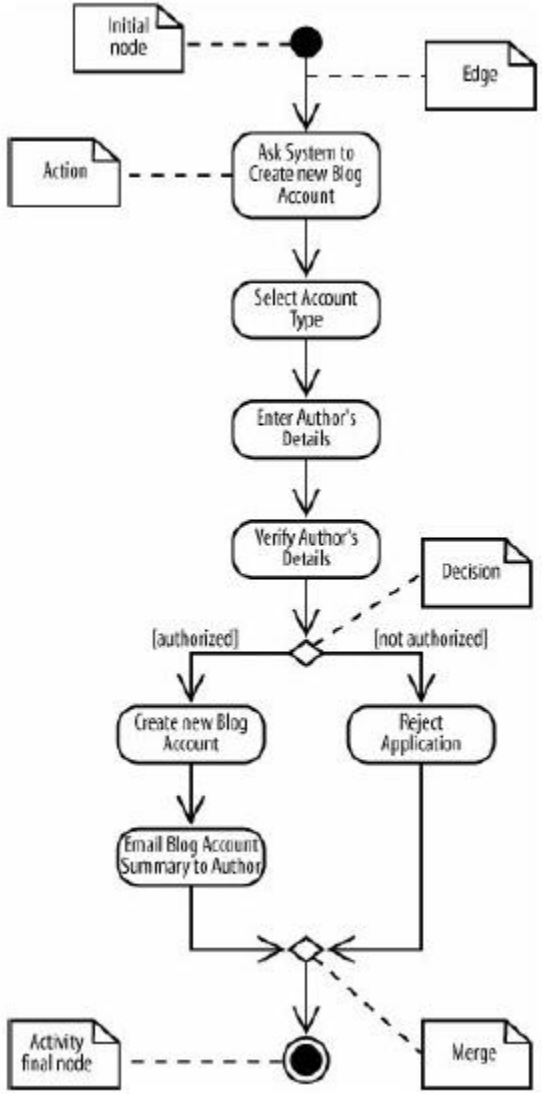
UML DIAGRAMS

- UML diagrams represent two different views of a system model:
  - Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
  - Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

UML DIAGRAMS



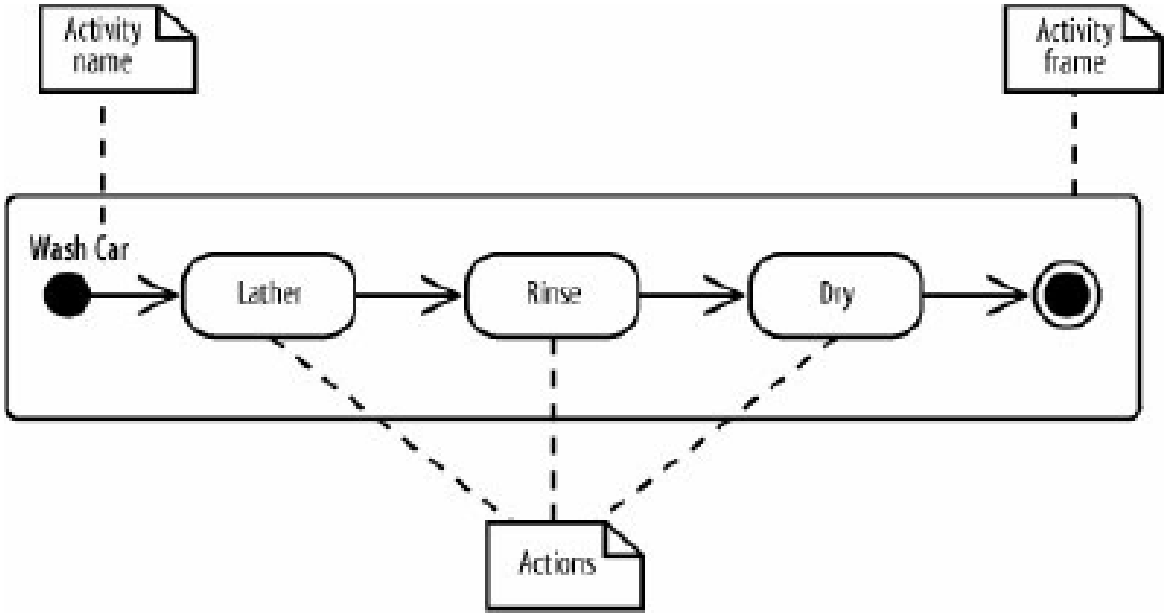
Activity diagram



Activity vs Action

- Activity is the "thing" being modeled **as a set of** actions.

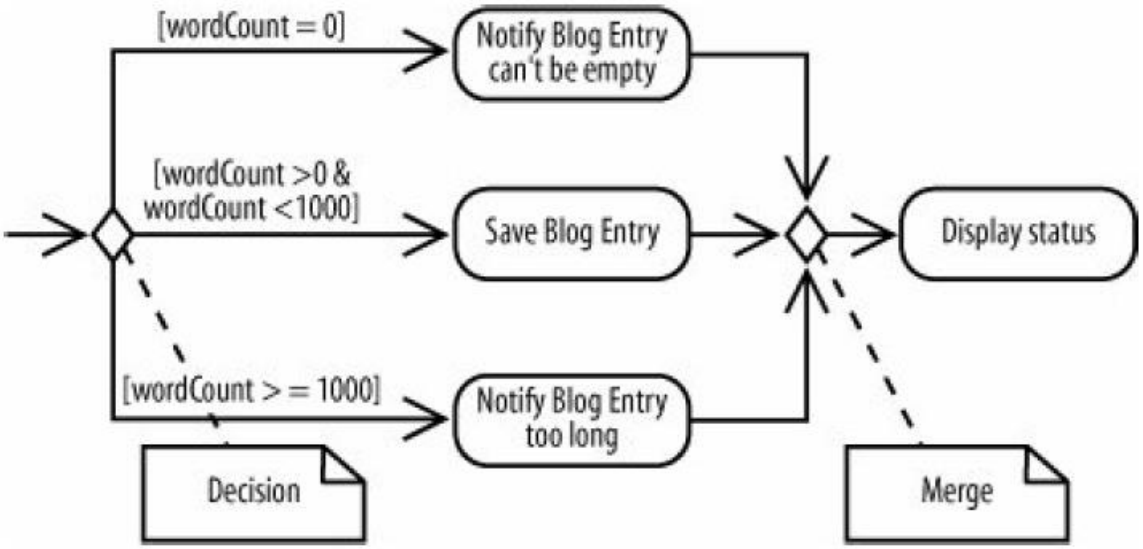
Figure 3-3. Capturing the three actionsLather, Rinse, and Drythat make up washing a car in an activity diagram



Decisions and guards

- Each branched edge contains a *guard condition* written in brackets. Guard conditions determine which edge is taken after a decision node.

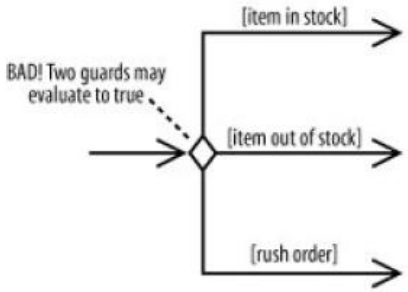
Figure 3-6. If the input value of age is 1200, then the Notify Blog Entry too long action is performed



Decisions and guards

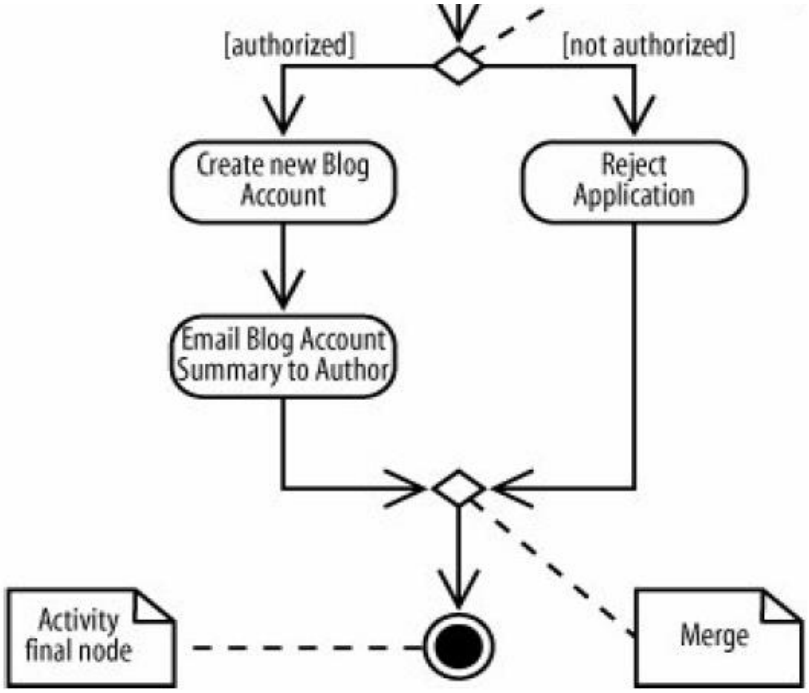
- Activity diagrams are clearest if the guards at decision nodes **are complete and mutually exclusive**. Figure 3-7 shows a situation in which the paths are not mutually exclusive.

Figure 3-7. Beware of diagrams where multiple guards evaluate to true



Merges

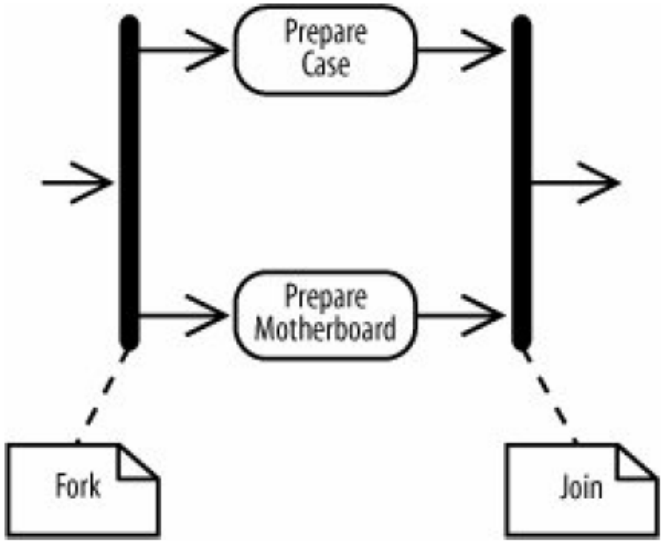
- As of UML 2.0, when multiple edges lead directly into an action, all incoming flows are waited on before proceeding (doing the action). But this doesn't make sense because only one edge is followed out of a decision node. You can avoid confusing your reader by explicitly showing merge nodes.



Forks

- You represent parallel actions in activity diagrams by using *forks* and *joins*, as shown in the activity diagram fragment in Figure 3-9.

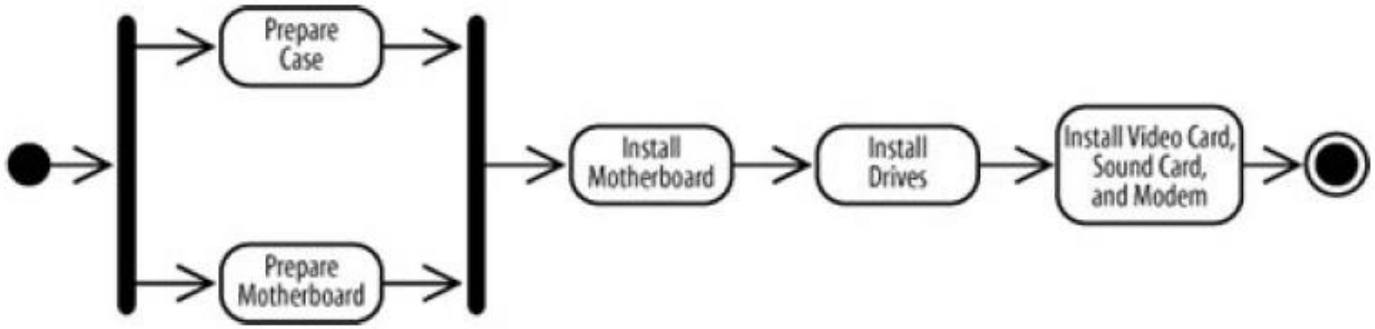
Figure 3-9. Both outgoing paths are followed at the fork, in contrast with decision nodes, where only one outgoing path is taken



JOINS

- When actions occur in parallel, it doesn't necessarily mean they will finish at the same time. In fact, one task will most likely finish before the other. However, the join prevents the flow from continuing past the join until all incoming flows are complete.

Figure 3-10. The computer assembly workflow demonstrates how forks and joins work in a complete activity diagram

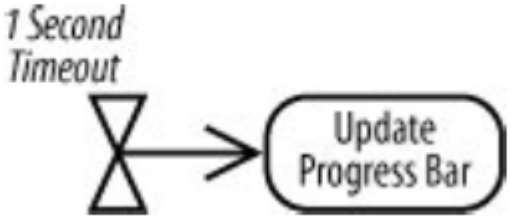


Time events

- Sometimes time is a factor in your activity. You may want to model a wait period, such as waiting three days after shipping an order to send a bill. You may also need to model processes that kick off at a regular time interval, such as a system backup that happens every week.

Figure 3-11. A time event with an incoming edge represents a timeout

Figure 3-12. A time event with no incoming flows models a repeating time event



Calling Other Actions

- As detail is added to your activity diagram, the diagram may become too big, or the same sequence of actions may occur more than once. When this happens, you can improve readability by providing details of an action in a separate diagram, allowing the higher level diagram to remain less cluttered.

Figure 3-13. Rather than cluttering up the top-level diagram with details of the Prepare Motherboard action, details are provided in another activity diagram

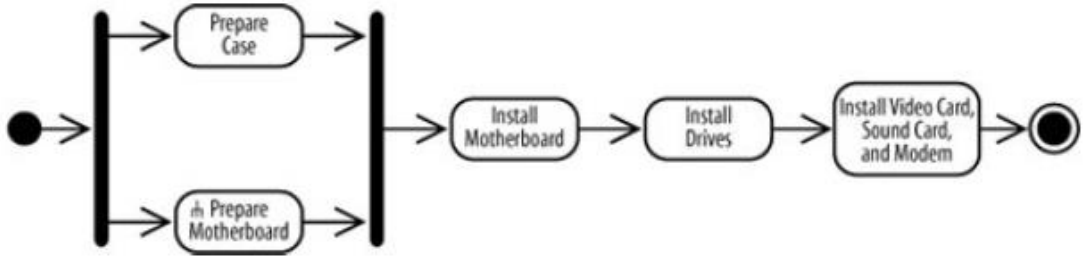


Figure 3-14. The Prepare Motherboard activity elaborates on the motherboard preparation process