Taming overly complex state-altering conditional logic

Dario Campagna

Head of Research and Development

State-altering conditional logic

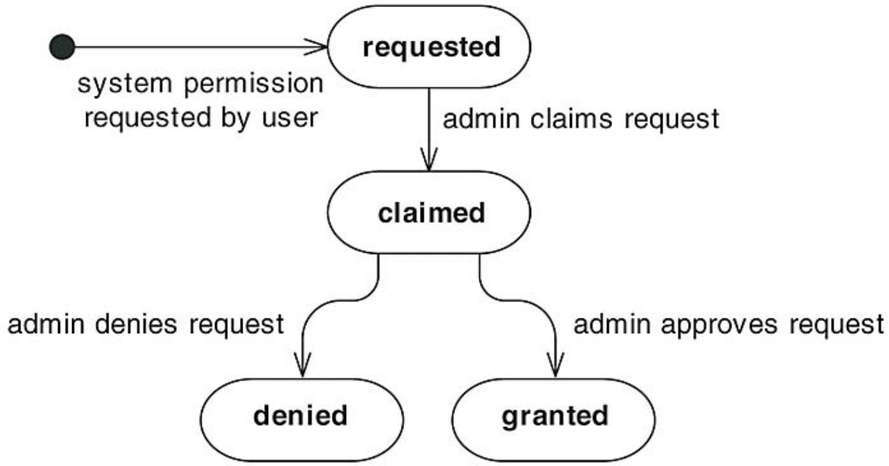
The conditional expressions that control an object's state transitions are complex.

- State-altering logic tends to spread.
- Logic can become hard to follow.
- Classes takes care of different responsibilities.
- Design gets complicated.

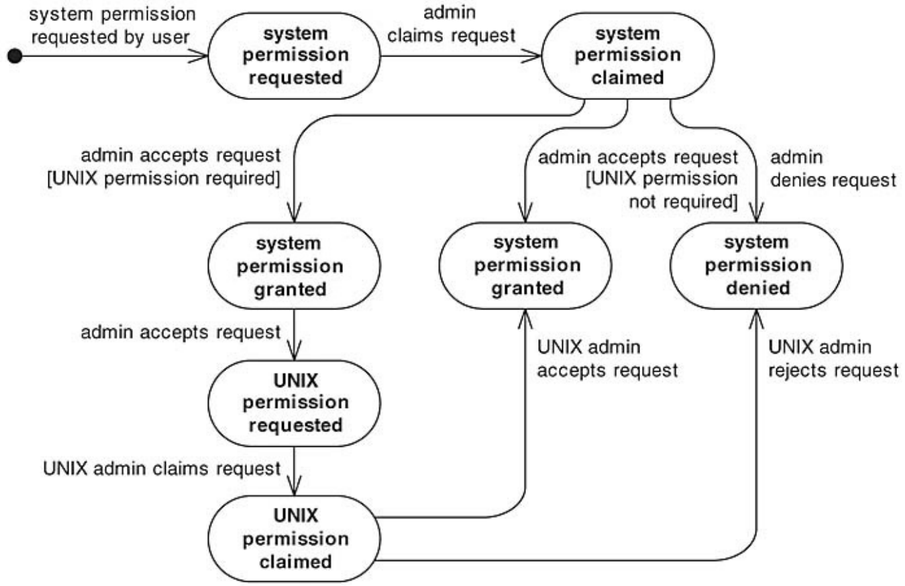
```
public class SystemPermission...
  private SystemProfile profile;
  private SystemUser requestor;
  private SystemAdmin admin;
  private boolean isGranted;
  private String state;

  public final static String REQUESTED = "REQUESTED";
  public final static String CLAIMED = "CLAIMED";
  public final static String GRANTED = "GRANTED";
  public final static String DENIED = "DENIED";

  public SystemPermission(SystemUser requestor, SystemProfile profile) {
    this.requestor = requestor;
    this.profile = profile;
    state = REQUESTED;
    isGranted = false;
    notifyAdminOfPermissionRequest();
  }

  public void claimedBy(SystemAdmin admin) {
    if (!state.equals(REQUESTED))
      return;
    willBeHandledBy(admin);
    state = CLAIMED;
  }

  public void deniedBy(SystemAdmin admin) {
    if (!state.equals(CLAIMED))
      return;
    if (!this.admin.equals(admin))
      return;
    isGranted = false;
    state = DENIED;
    notifyUserOfPermissionRequestResult();
  }

  public void grantedBy(SystemAdmin admin) {
    if (!state.equals(CLAIMED))
      return;
    if (!this.admin.equals(admin))
      return;
    state = GRANTED;
    isGranted = true;
    notifyUserOfPermissionRequestResult();
  }
}
```

State-altering conditional logic

State-altering conditional logic can quickly become hard to follow as more real-world behavior gets added.



A state diagram that can be managed with not very complicated conditional logic (see previous slide).



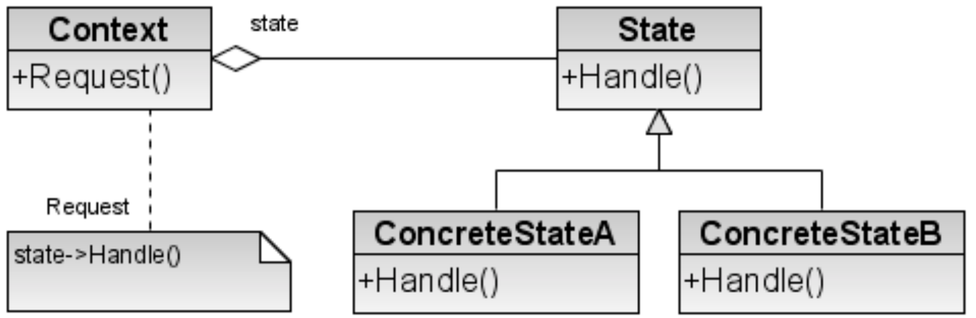
Conditional logic shows its limit when the state transition logic is more elaborated.

State

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Motivation

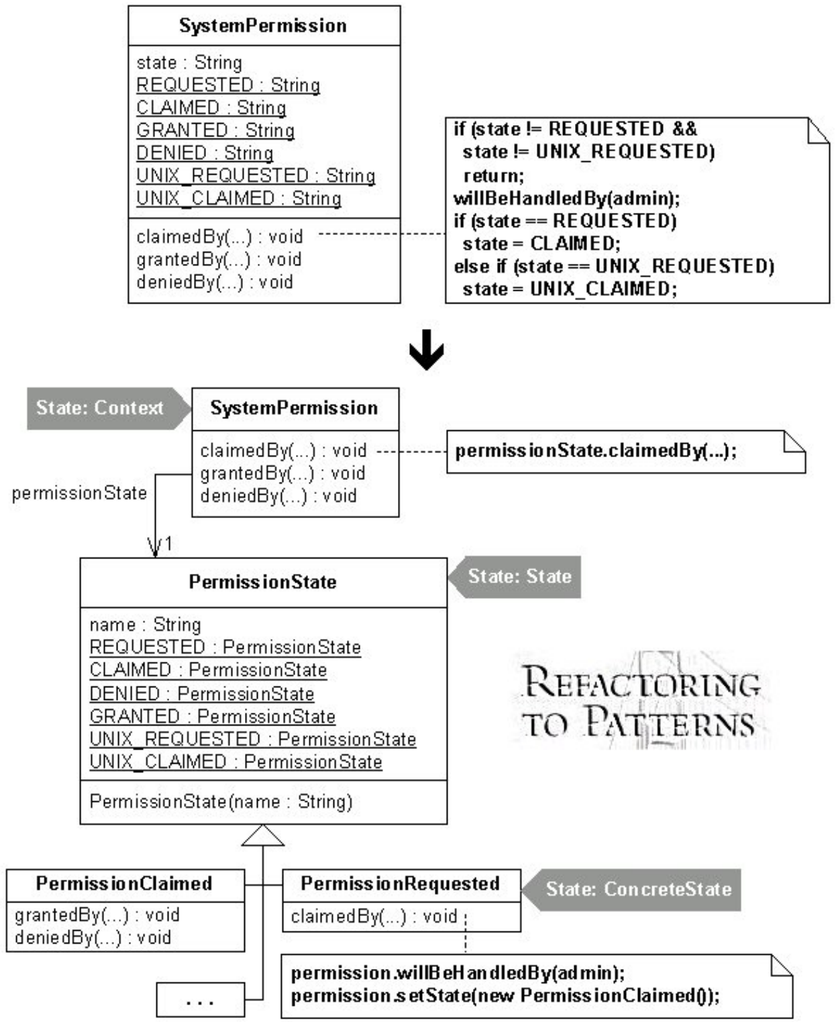
- A TCPConnection class that responds differently based on its state
- To have a good bird-eye view of state-changing logic



Applicability

- An object's behavior depends on its state, and it must change at runtime
- Operations have complicated conditional statements depending on the object's state

Replace State-Altering Conditionals with State



https://www.industriallogic.com/xp/refactoring/alteringConditionalsWithState.html
https://www.informit.com/articles/article.aspx?p=1398607&seqNum=4

Replace State-Altering Conditionals with State

Benefits	Liabilities
Reduces or removes state-changing conditional logic.	Complicates a design when state transition logic is already easy to follow.
Simplifies complex state-changing logic.	
Provides a good bird’s-eye view of state-changing logic.	

Replace State-Altering Conditionals with State – Mechanics

1. Apply *Replace Type Code with Class* on the original state field, the new class is the *state superclass*.

✓ Compile

2. Apply *Extract Subclass* to each state constants. Declare the state superclass to be abstract.

✓ Compile

3. Copy a context class method that alters the state to the state superclass, add delegation call in superclass. (For every context method)

✓ Compile and test

4. Copy state superclass methods altering a state to the corresponding state subclass, remove unrelated logic. (For all the states)

✓ Compile and test

5. Delete the body of each methods moved to the state superclass in step 3.

✓ Compile and test

Replace State-Altering Conditionals with State – Example

Let's apply this refactoring to the `SystemPermission` class in the *state-altering-conditionals* branch of the following repository.

https://github.com/dario-campagna/replace-state-altering-conditionals-with-state

- Example from Refactoring to Patterns
- Code comes from a security system