Rules Pattern

Steve Smith

Ardalis.com

Twitter: @ardalis

pluralsight
hardcore developer training

Related Courses

SOLID Principles of Object Oriented Design

This course introduces foundational principles of creating well-crafted code and is appropriate for anyone hoping to improve as a developer

Motivating Example

Rules
Pattern

- **A class or method has complex and growing business logic**
- **Additional changes of the same nature are likely**
- **Examples:**
  - Customer discount calculations
  - Social gamification rules (badges/points)
  - Credit / Insurance rating

Demo

Customer Discounts Example

Demo: Customer Discounts

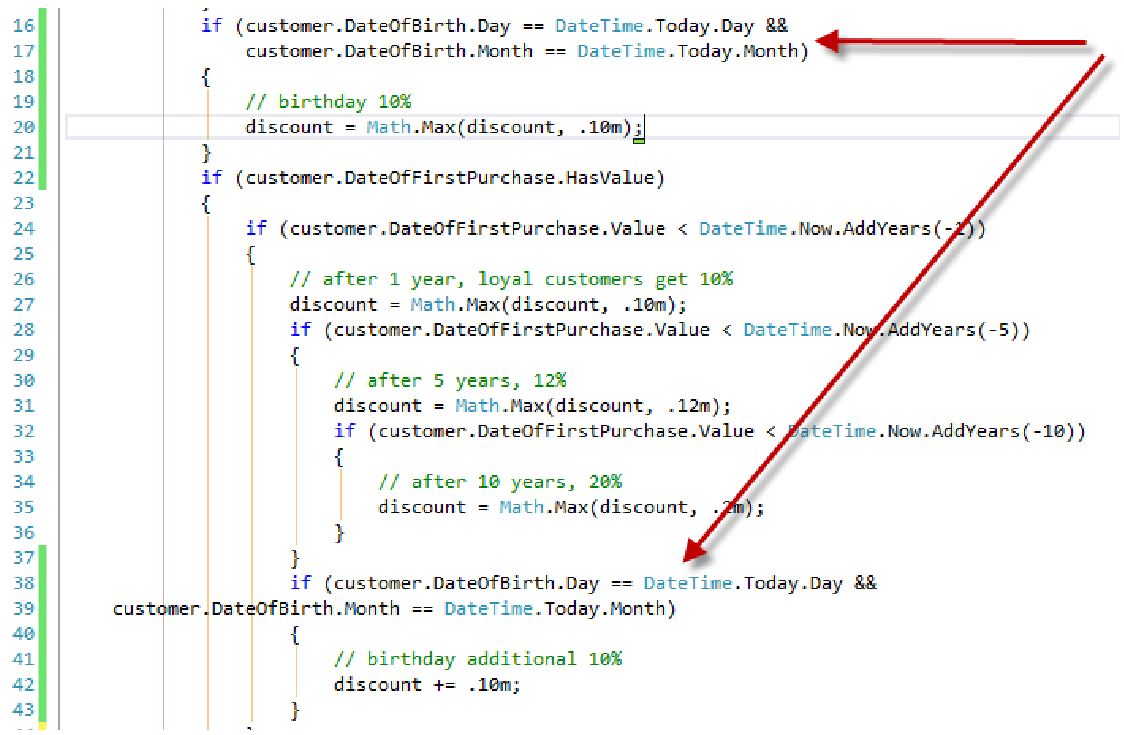
- **First-Time Buyers get 15%**
- **Veterans get 10%**
- **Loyal customers get**
  - 10% if they’ve been a customer for a year
  - 12% if they’ve been a customer for 5 years
  - 20% if they’ve been a customer for 10 years
- **Seniors get 5%**

- **Customers always receive the best discount that applies**

- **Marketing just started a new promotion:**
  - Customers get 10% off on their birthday
  - Loyal customers get an extra 10% off on their birthday

What's the Problem?

- **Growing Complexity**
  - Cyclomatic complexity just grew from 7 to 11
  - http://bit.ly/ieyeRy (more on Cyclomatic Complexity)
- **Duplicate Code**



```
        if (customer.DateOfBirth.Day == DateTime.Today.Day &&
            customer.DateOfBirth.Month == DateTime.Today.Month)
        {
            // birthday 10%
            discount = Math.Max(discount, .10m);
        }
        if (customer.DateOfFirstPurchase.HasValue)
        {
            if (customer.DateOfFirstPurchase.Value < DateTime.Now.AddYears(-1))
            {
                // after 1 year, loyal customers get 10%
                discount = Math.Max(discount, .10m);
                if (customer.DateOfFirstPurchase.Value < DateTime.Now.AddYears(-5))
                {
                    // after 5 years, 12%
                    discount = Math.Max(discount, .12m);
                    if (customer.DateOfFirstPurchase.Value < DateTime.Now.AddYears(-10))
                    {
                        // after 10 years, 20%
                        discount = Math.Max(discount, .2m);
                    }
                }
                if (customer.DateOfBirth.Day == DateTime.Today.Day &&
customer.DateOfBirth.Month == DateTime.Today.Month)
                {
                    // birthday additional 10%
                    discount += .10m;
                }
```

Intent

Rules Pattern

- Separate individual rules from rules processing logic
- Allow new rules to be added without the need for changes in the rest of the system

Single Responsibility Principle

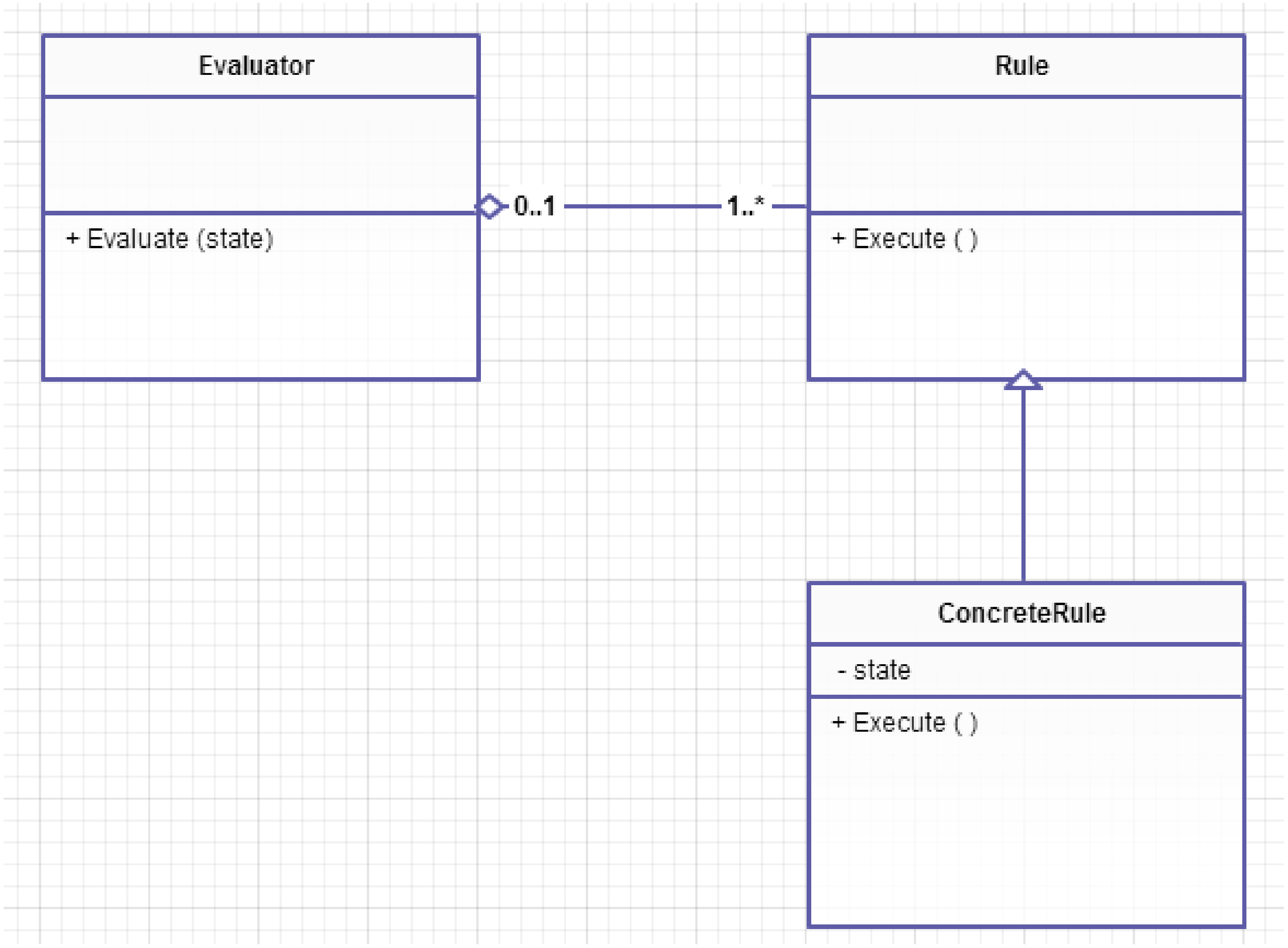
A class or method should only have one reason to change.

Applicability

Consider using the Rules Pattern when:

- a system is suffering from *conditional complexity*, and additional changes of the same nature are anticipated
- a system has comingled the concerns of choosing which action(s) are applicable, and executing these actions
- a system needs to support user-created logic for determining when and how to apply actions

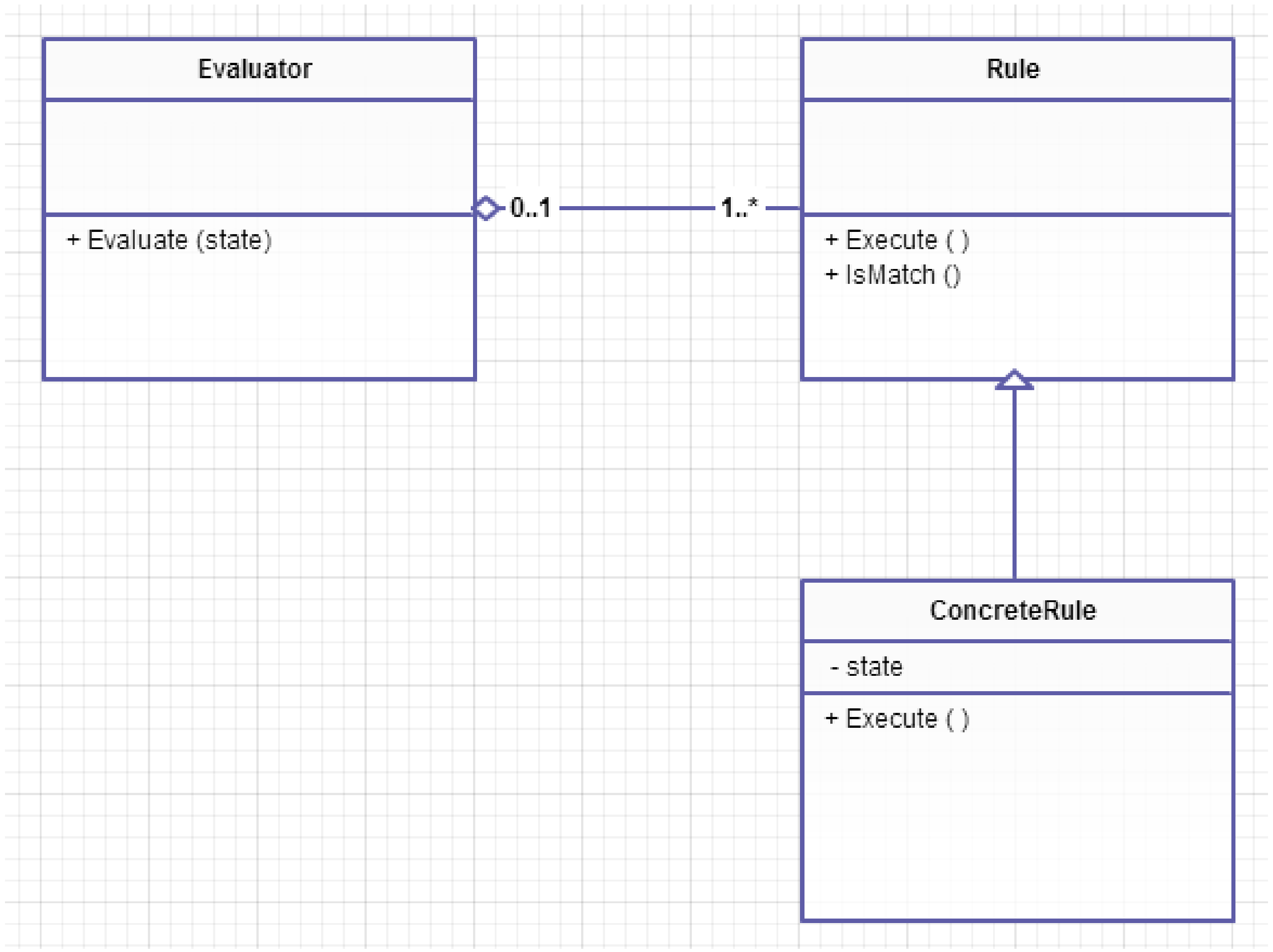
Structure


Evaluator
+ Evaluate (state)
0..1
1..*
Rule
+ Execute ( )
ConcreteRule
- state
+ Execute ( )

Demo

Customer Discounts Rules Engine

Structure


Evaluator
+ Evaluate (state)
0..1
1..*
Rule
+ Execute ( )
+ IsMatch ()
ConcreteRule
- state
+ Execute ( )

Rule Considerations

Read Only?

Dependencies

Explicit Order

Priority

Persistence

User Interface

HALT!

Business Rules Engines

Rules Pattern

- Software Systems Designed to Encapsulate Business Rules
- Typically support authoring of rules by business users
- Rules stored in a database or filesystem
- Many commercial and open source options
- .NET 3+ includes a Rules Engine in System.Workflow
  - Learn more: http://bit.ly/aetfj0

Practice: The Greed Game Kata

- **Write a Greed game scoring method using the rules in the kata**
  - In the before folder for this module's files
  - Online: http://nimblepros.com/media/36619/greed%20kata.pdf
- **Once you encounter conditional complexity, refactor to use the rules pattern**
- **Compare your solution with others online**
- **Repeat the exercise until you're comfortable applying the Rules Pattern**

Summary

- Consider using the Rules Pattern when you have a growing amount of conditional complexity
- Separate the logic of each individual rule and its effects into its own class
- Separate the selection and processing of rules into a separate Evaluator class
- Consider using a Business Rules Engine application or component if your application's requirements warrant it

References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Online:

Soft Coding: http://thedailywtf.com/articles/soft_coding.aspx

Should You Use a Rules Engine?: http://www.jessrules.com/guidelines.shtml

Simple .NET Rules Engine Discussion (StackOverflow): http://bit.ly/fDH8r

Business Rules Engines: http://en.wikipedia.org/wiki/Business_rules_engine

Intro to WF Rules Engine: http://bit.ly/aetfj0

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice

pluralsight

hardcore developer training