Strategy Pattern

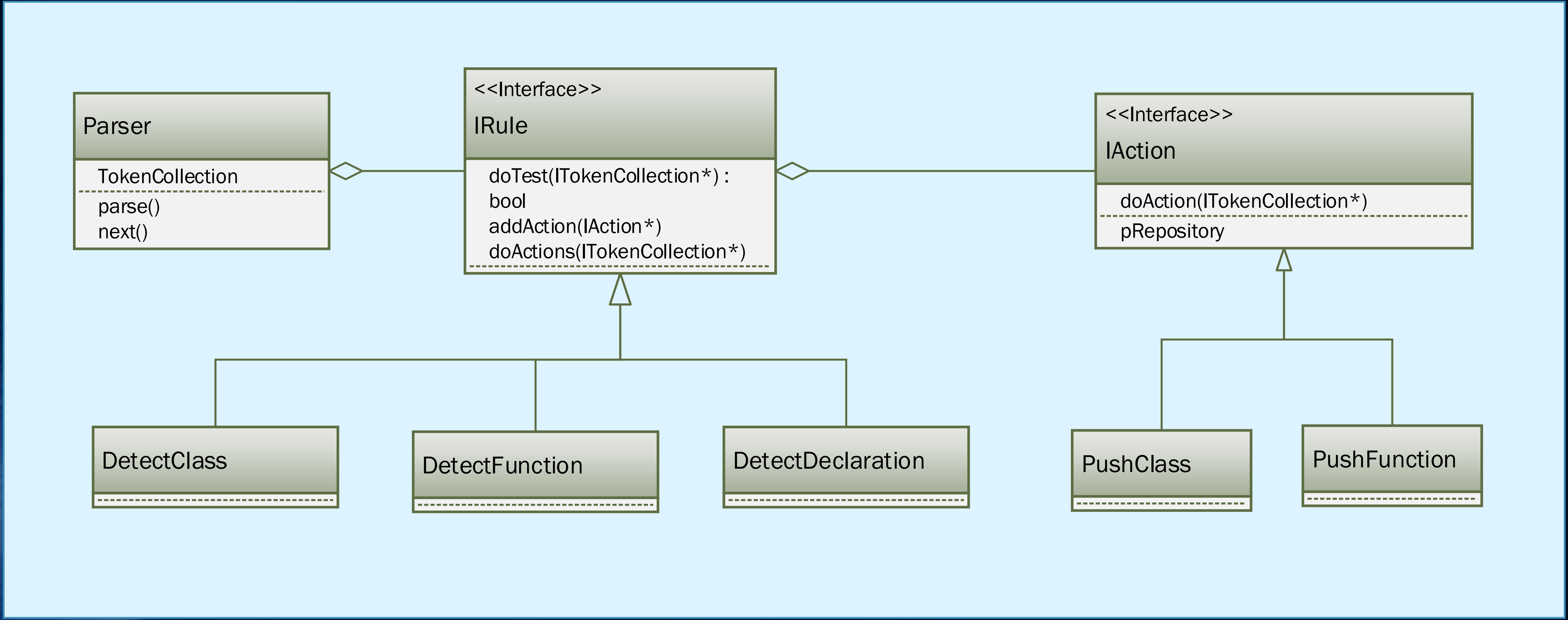
Jim Fawcett
CSE776 – Design Patterns
Fall 2014

Motivation

Code analysis requires parsing of code text that entails detection of specific language constructs and storing results. Placing the detection and storage in one class isn’t desirable for several reasons:

- Each grammatical construct needs a specific rule for detection and there are many such rules.
- The actions required after detection depend on which rule matches the current context.
- Actions may depend on the results of the actions of other rules.

Parser uses Strategy

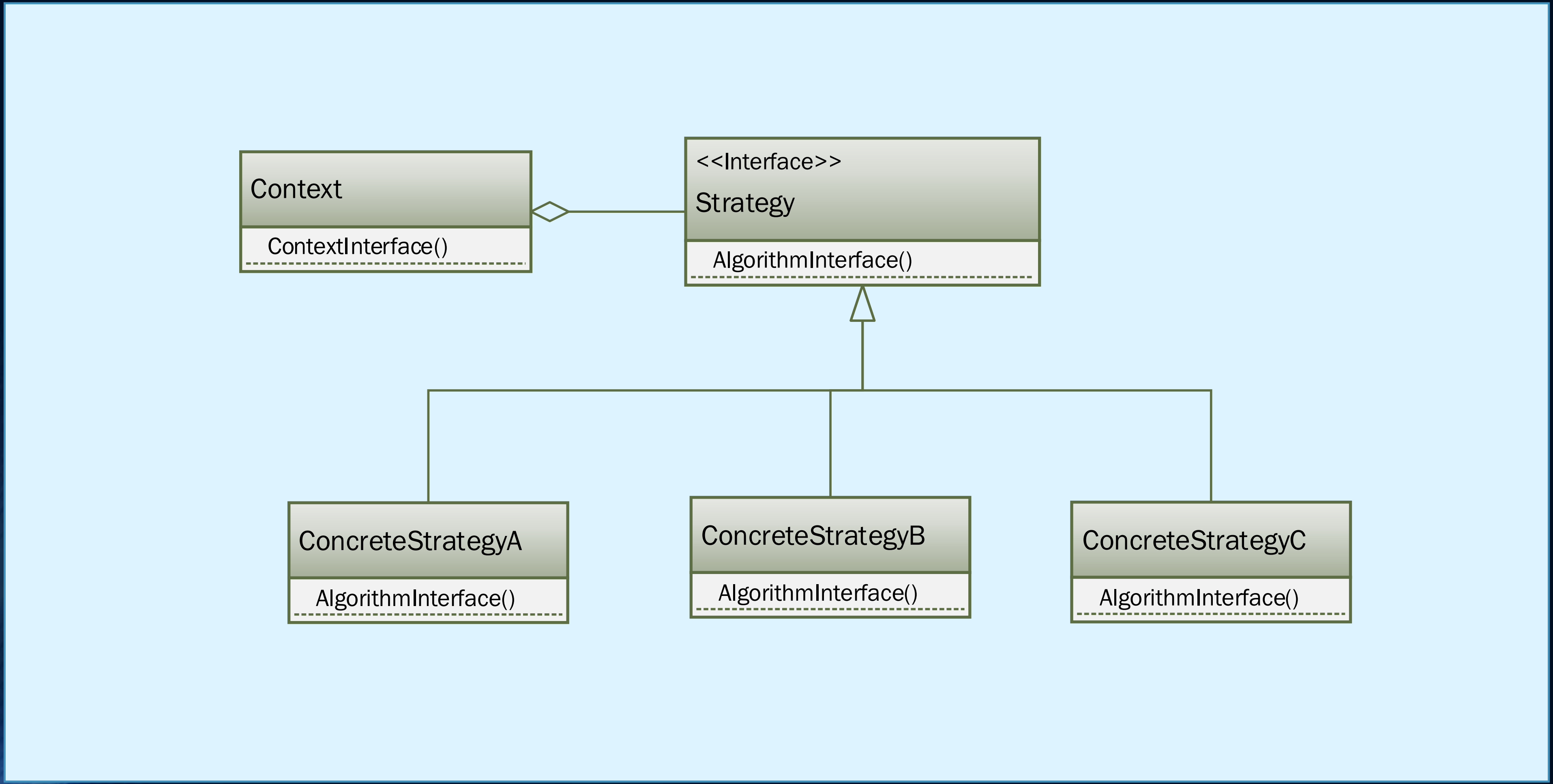


Applicability

Use the Strategy Pattern where:

- Many related classes differ only in their behavior (rules, actions)
- You need variants of an algorithm (rules, actions)
- An algorithm uses data a client shouldn’t know about (Token Collections)
- A class defines many behaviors and these appear as multiple conditional statements in its operations. Instead, move related conditional branches to their own Strategy class.

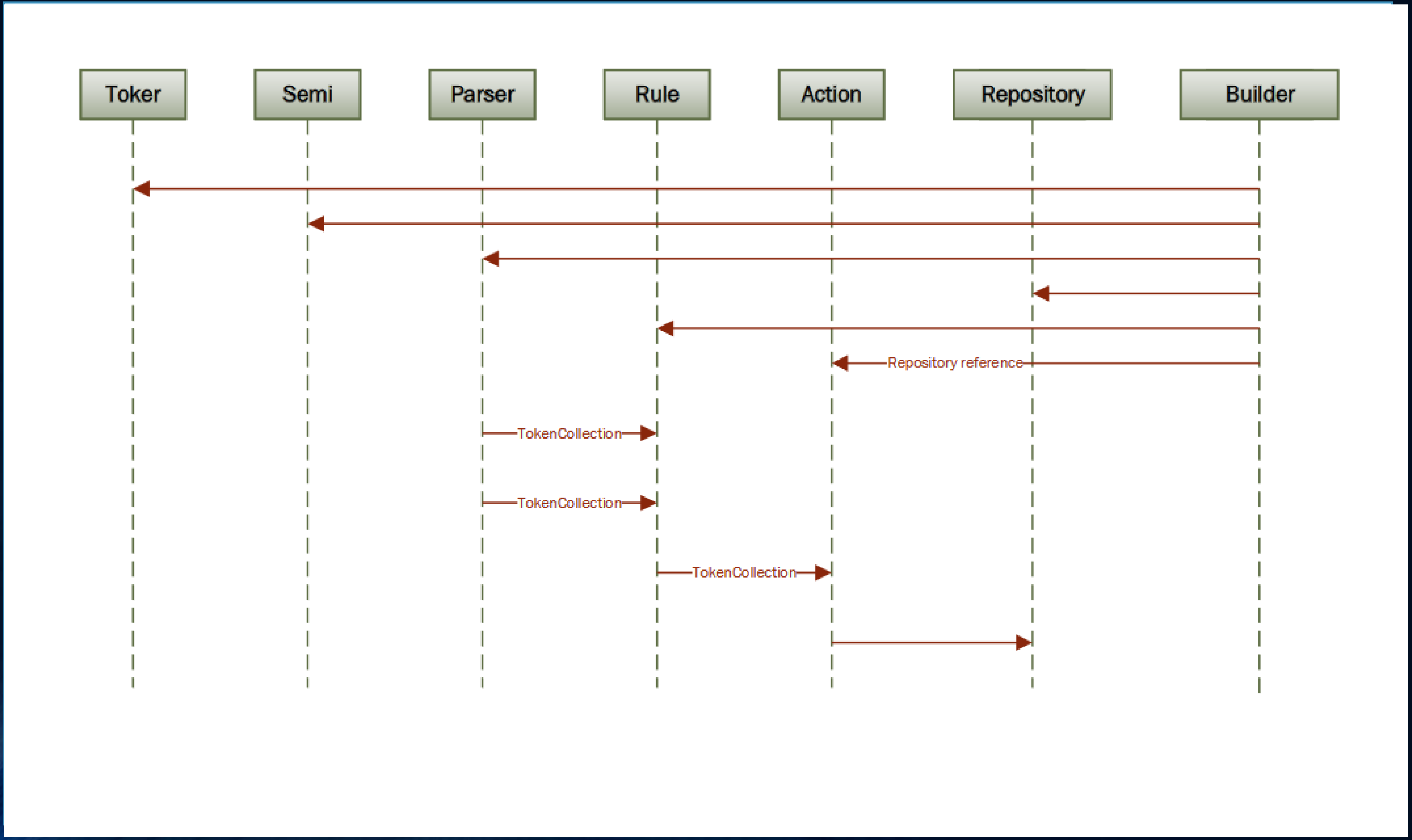
Structure


Context
ContextInterface()
<<Interface>>
Strategy
AlgorithmInterface()
ConcreteStrategyA
AlgorithmInterface()
ConcreteStrategyB
AlgorithmInterface()
ConcreteStrategyC
AlgorithmInterface()

Participants

- Strategy (IRule, IAction)
  - Declares interface common to all ConcreteStrategies
  - Context uses this interface to call the algorithm defined by a ConcreteStrategy
- ConcreteStrategy (DetectClass, PushClass)
  - Implements the algorithm using the Strategy interface
- Context (Parser for rules, Rule for actions, Repository)
  - Is configured with Concrete Strategy objects (rules, actions)
  - Maintains references to Strategy objects (Builder maintains references)
  - May define an interface that lets Strategy access its data (Repository)

Collaborations



Collaborations

- Strategy (Rule, Action) and Context (Parser) interact to implement the chosen algorithm.
- A Context (Parser) may pass all data required by the algorithm (TokenCollection) to the Strategy (Rule) when the algorithm is called.
- Alternately the Context may pass a reference to itself to the Strategy. That lets the Strategy call back on the Context as required.
- A Context forwards requests from its clients to its Strategy. Clients usually create and pass a ConcreteStrategy object to the Context (Builder); Thereafter clients interact exclusively with the Context.

Consequences

- The Strategy Pattern has the following benefits and drawbacks:
  - Hierarchies of Strategy classes define a family of algorithms or behaviors for Contexts to reuse. Inheritance can help to factor out common functionality of the algorithms.
  - Encapsulating the algorithm in separate Strategy classes lets you vary the algorithm independently of its Context, making it easier to switch, understand, and extend.
  - Strategies eliminate large sets of conditional statements.
  - Clients must be aware of different Strategies (can use Builder to mitigate)
  - Communication overhead between Strategy and Context
  - Increased number of objects

Implementation

- The Strategy and Context interfaces must give a ConcreteStrategy efficient access to data it needs from the Context.
- Strategies as template parameters
  - Templates can be used to configure a class with a Strategy. This makes sense when only one Strategy, selected by the client, is used for the duration of the processing of an algorithm. Sorting is one example.

Code Examples

- Flight Strategy
  - StrategyPatternCode
- Sorting Strategy

Know Uses

- Parsers used in CSE681 – SMA and CSE687 – OOD
- Control systems where the processing for an operation depends on the operation’s context, e.g., flight navigation control.
- Games where player’s may choose one or more strategies to achieve some objective.
- Software tools like Code Analyzers that use parsers.
- Graphical processors that build parse trees, e.g., browser and its HTML parse tree

Related Patterns

- Decorator
  - Both Decorator and Strategy can dynamically add new behaviors
- Template Method
  - Template method provides algorithm steps defined by a template
- Composite
  - Both Composite and Strategy build processing out of predefined parts

End of Presentation