EXTENSION OBJECT

Adarsh
Mrunal
Prof. Fawcett

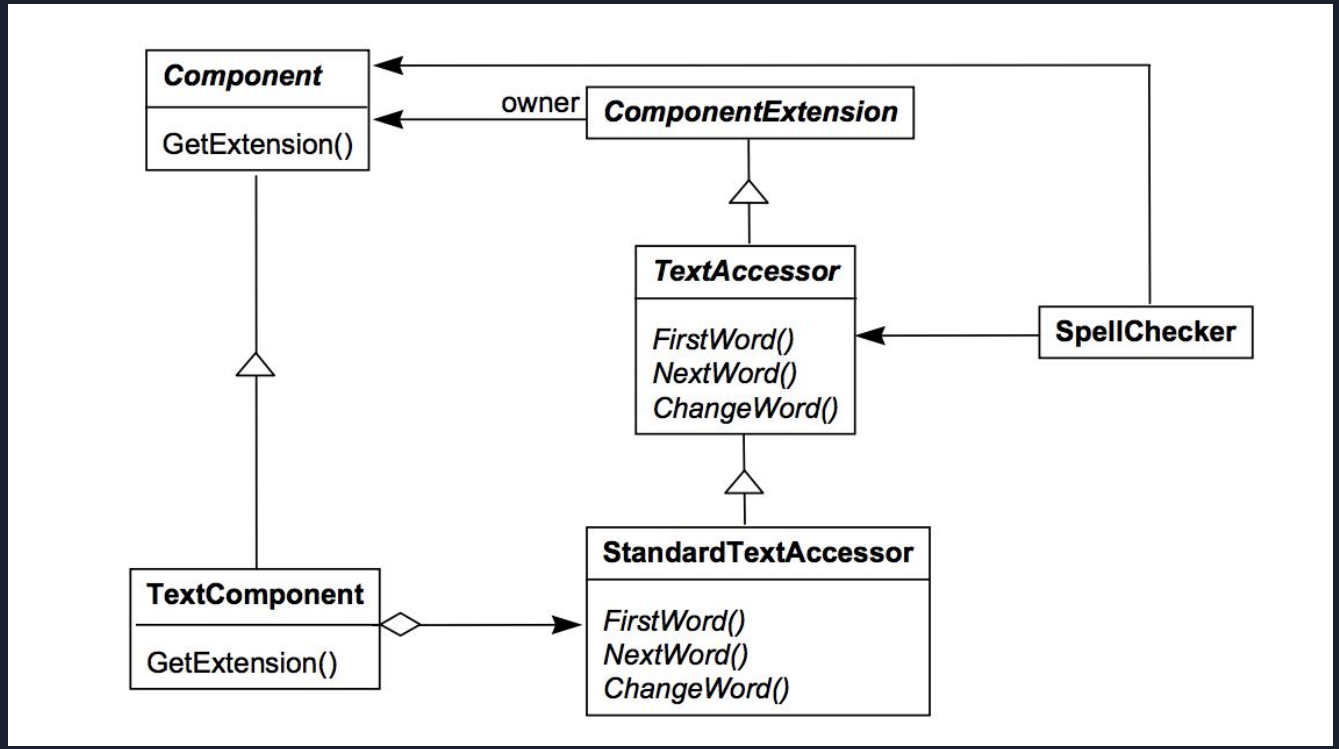
INTRODUCTION

- Extension object is a behavioural design pattern
- INTENT
  - Anticipate that an object’s interface needs to be extended in the future. Additional interfaces are defined by extension objects.

MOTIVATION

- For some abstractions it is difficult to anticipate their complete interface since different clients can require a different view on the abstraction
- Avoid having bloated interface.

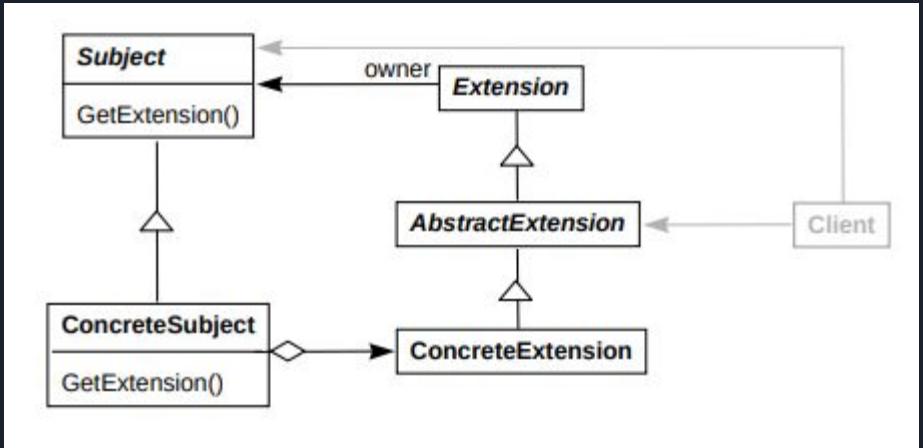
MOTIVATION EXAMPLE



APPLICABILITY

- Addition of new or unforeseen interfaces to existing classes without impacting all clients
- Class representing a key abstraction plays different roles for different clients.
- Class should be extensible with new behavior without subclassing from it

BASIC STRUCTURE



PARTICIPANTS

- Subject (Component)
  - Defines the identity of an abstraction
- Extension (ComponentExtension)
  - It defines some support for managing extensions themselves
- ConcreteSubject (StandardTextComponent)
  - Implement the GetExtension operation to return a corresponding extension object when the client asks for it
- AbstractExtension (TextAccessor)
  - Declares the interface for a specific extension
- ConcreteExtension (StandardTextAccessor)
  - Implement the extension interface for a particular component

COLLABORATORS

- A client asks a Subject for a specific extension
- When the extension exists the Subject returns a corresponding extension object
- If the Subject doesn't support an extension it returns nil to signal that it doesn't support it.

IMPLEMENTATION

A Subject class would be declared like this in C++:

```
class Subject {
public:
   //...
   virtual Extension* GetExtension(const char* name);
};
```

Subject::GetExtension is implemented as:

```
Extension* Subject::GetExtension(const char* name)
{
   return 0;
}
```

Here is a ConcreteSubject that provides a SpecificExtension:

```
class ConcreteSubject: public Subject {
public:
   //...
   virtual Extension* GetExtension(const char* name);
private:
   SpecificExtension* specificExtension;
};
```

The implementation of ConcreteSubject::GetExtension is defined like:

```
Extension* ConcreteSubject::GetExtension(const char* name)
{
   if (strcmp(name, "SpecificExtension" == 0) {
      if (specificExtension == 0)
         specificExtension = new SpecificExtension(this);


      return specificExtension;
   }
   return Subject::GetExtension(name);
}
```

Finally, to access an extension the client writes:

```
SpecificExtension* extension;
Subject* subject;

extension = dynamic_cast<SpecificExtension*>(
                  subject->GetExtension("SpecificExtension")
                     );
if (extension) {
   // use the extension interface
}
```

CONSEQUENCES

- Extension Objects facilitates adding interfaces
- No bloated class interfaces for key abstractions.
- Support for modeling different roles of a key abstraction in different subsystems
- Clients become more complex
- Tension to abuse extensions for concepts that should be explicitly modeled

LIABILITIES

- Internal vs. External extensions
- Identifying extensions.
- Demand loading of extensions
- Freeing Extension Objects

KNOWN USES

- OpenDoc
- OLE
- UI framework of Taligent operating system

RELATED PATTERNS

- Visitor
- Adapter
- Decorator

REFERENCES

- https://ecs.syr.edu/faculty/fawcett/handouts/CSE776/PatternPDFs/ExtensionObject.pdf

THANK YOU