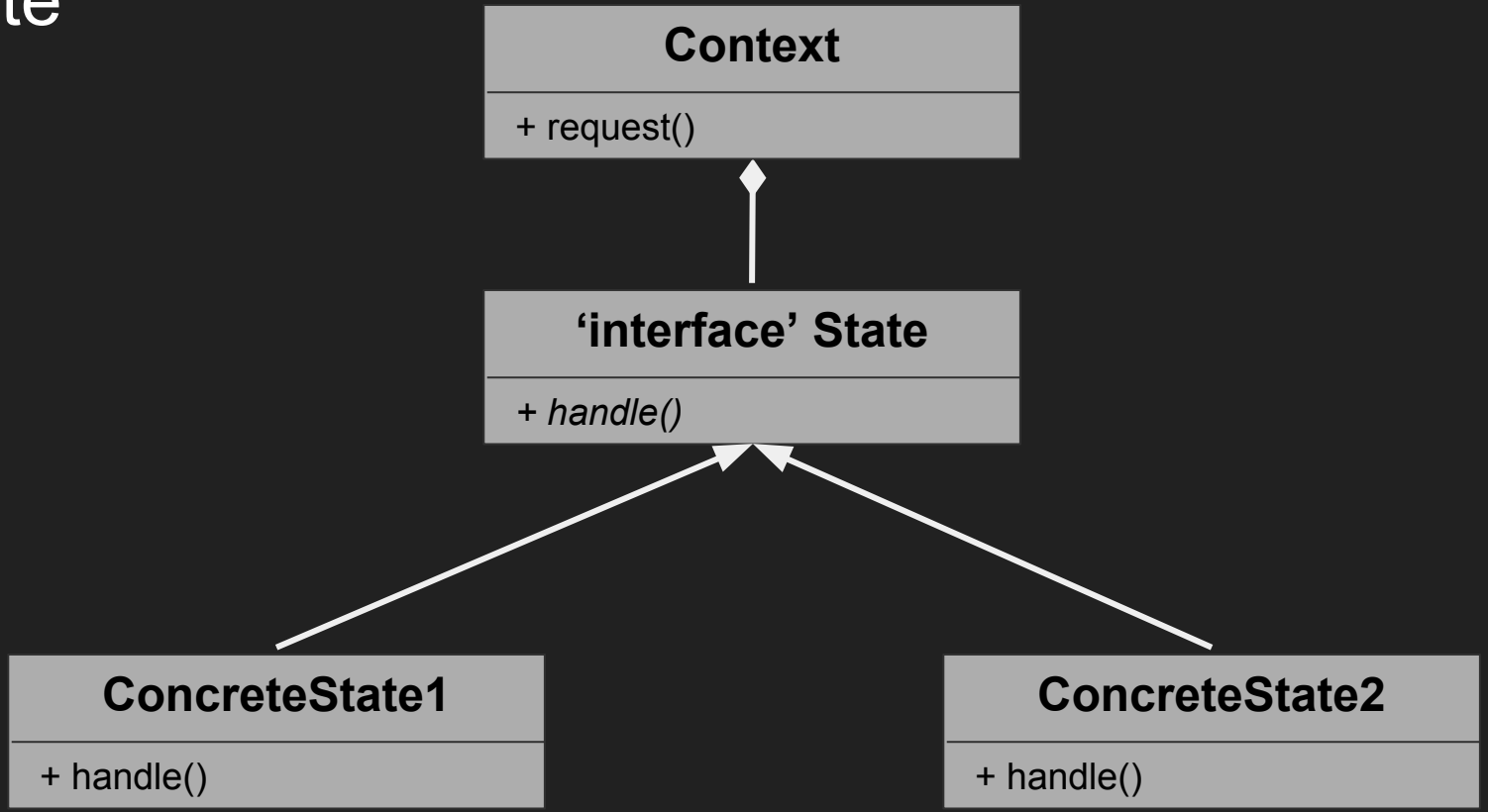
Технологии программирования

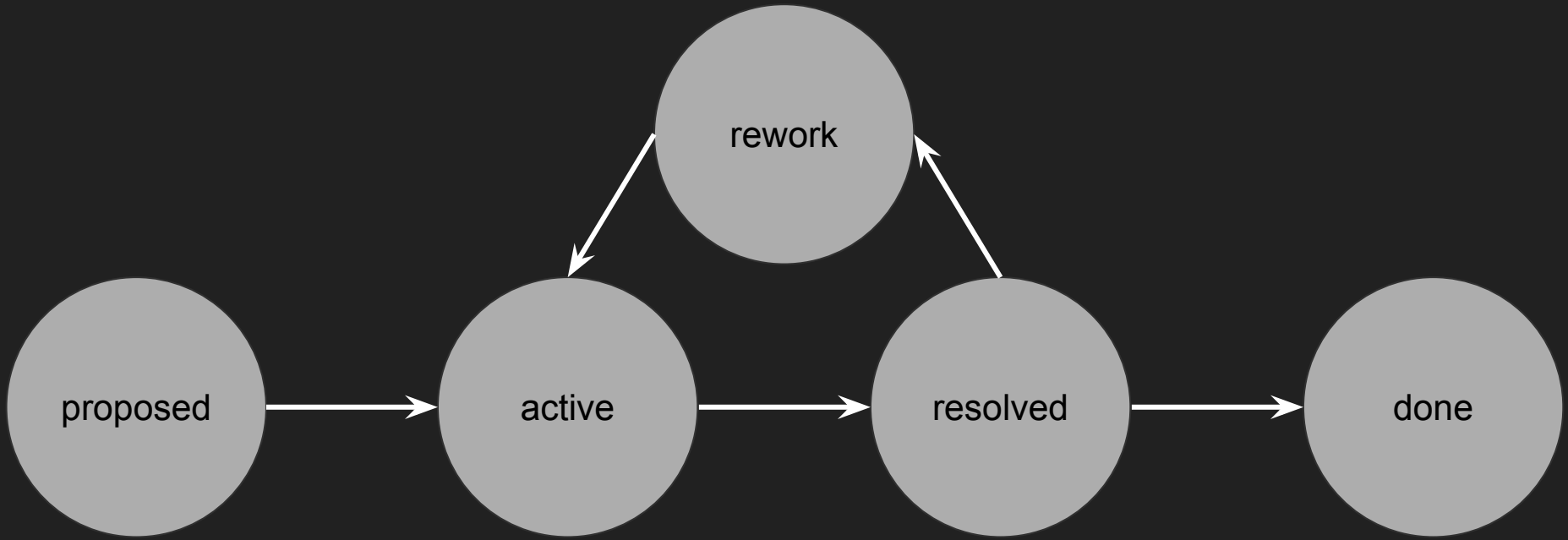
Поведенческие паттерны. State, Visitor, Observer

Александр Михалевич

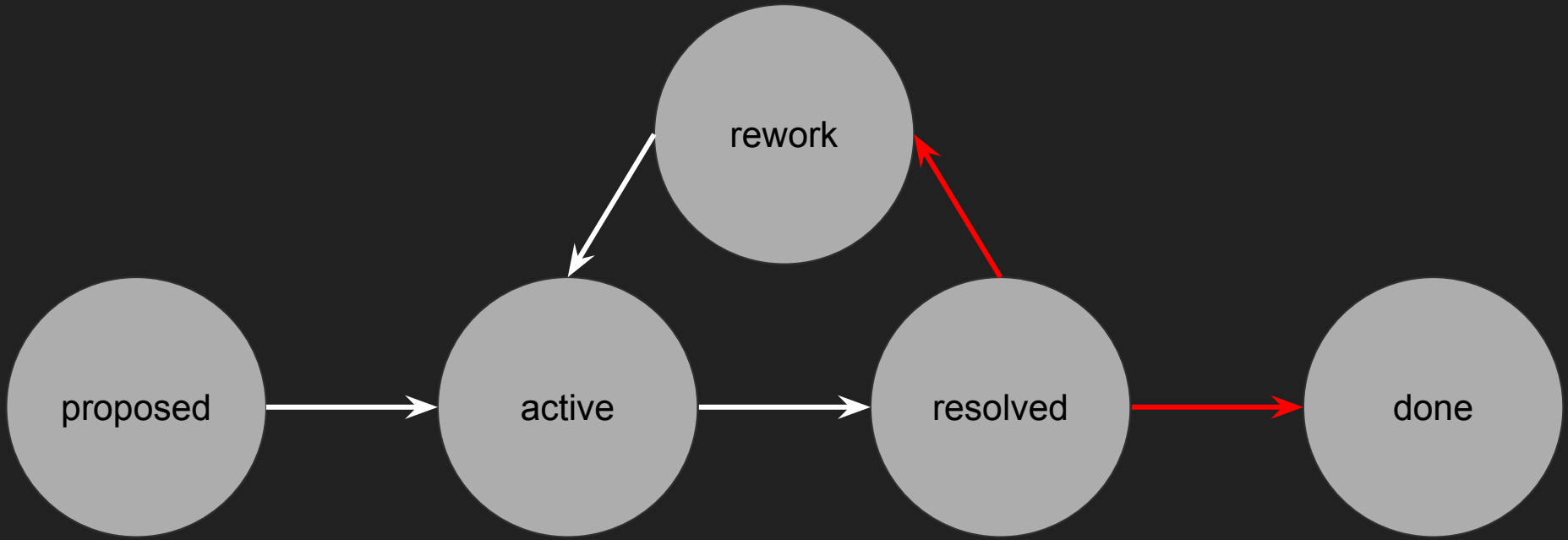
State


Context
+ request()
‘interface’ State
+ handle()
ConcreteState1
+ handle()
ConcreteState2
+ handle()

State example



State example

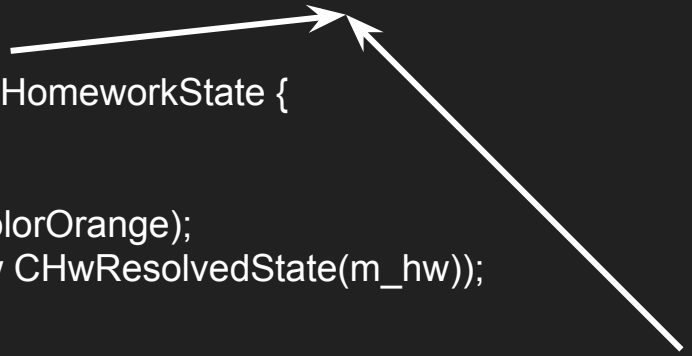


State example

```
class CHomeworkState {
protected:
	CHwElement* m_hw;
public:
	CHomeworkState(CHomework* hw) : m_hw(hw) {}
	virtual void set_active() {}
	virtual void set_resolved() {}
	virtual void set_rework() {}
	virtual void set_done() {}
};
```

State example

```
class CHomeworkState;
```



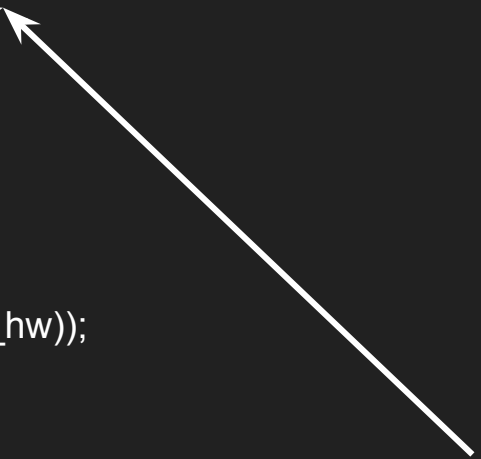
```
class CHwActiveState : public CHomeworkState {
public:
        void set_resolved() {
                m_hw.set_color(eColorOrange);
                m_hw.set_state(new CHwResolvedState(m_hw));
        }
};
```

```
class CHwResolvedState : public CHomeworkState {
public:
        void set_rework() {
                m_hw.set_color(eColorRed);
                m_hw.set_state(new CHwReworkState(m_hw));
        }
        void set_done() {
                m_hw.set_color(eColorGreen);
                m_hw.set_state(new CHwDoneState(m_hw));
        }
};
```

State example

```
class CHwProposedState : public CHomeworkState {
public:
        void set_active() {
                m_hw.set_color(eColorBlue);
                m_hw.set_state(new CHwActiveState(m_hw));
        }
};
```

```
class CHomeworkState;
```



```
class CHwReworkState : public CHomeworkState {
public:
        void set_active() {
                m_hw.set_color(eColorBlue);
                m_hw.set_state(new CHwActiveState(m_hw));
        }
};
```

```
class CHwDoneState : public CHomeworkState {};
```

State example

```
// client code

class CHwElement {
private:
        CHomeworkState m_state;
        const std::string m_caption;
        const std::string m_description;
public:
        CHwElement(const std::string& caption, const std::string& description) :
                m_caption(caption), m_description(description), m_state(CHwProposedState()) {}
        void set_state(const CHomeworkState& state) { m_state = state; }
};
```

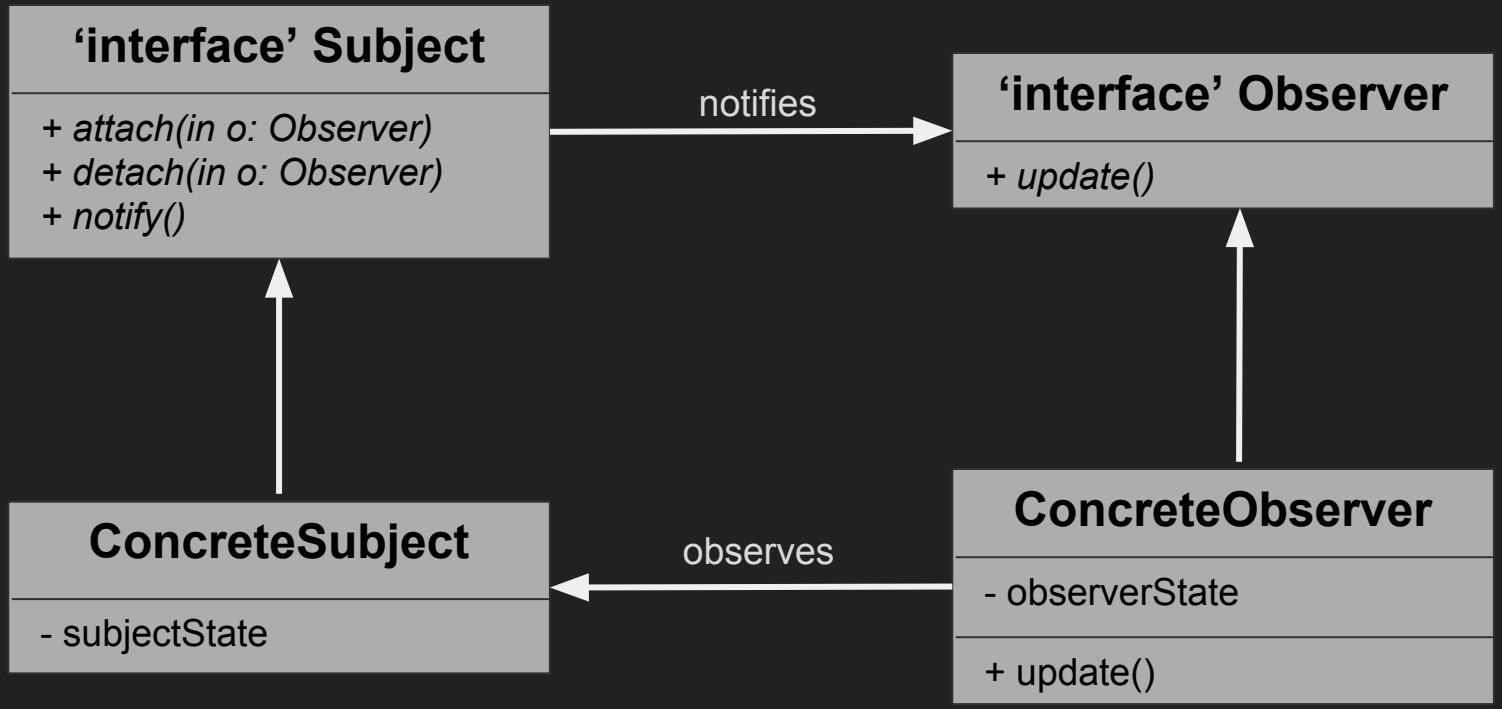
Relations

State vs Strategy

Когда применять

необходимо реализовать
конечный автомат
(машину состояний)

Observer

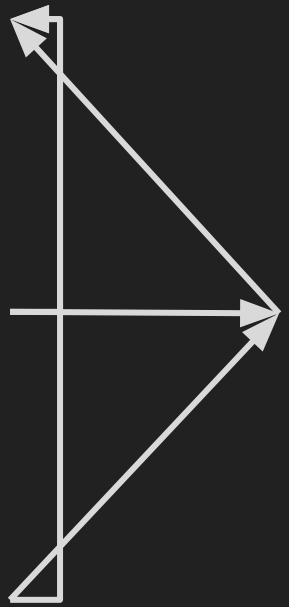

‘interface’ Subject
+ attach(in o: Observer)
+ detach(in o: Observer)
+ notify()
notifies
‘interface’ Observer
+ update()
ConcreteSubject
- subjectState
observes
ConcreteObserver
- observerState
+ update()

Observer example

```
class IObserver {
public:
        virtual ~IObserver() {}
        virtual void handle(DatabaseManager* db) = 0;
};
```

```
class UsersObserver : public IObserver {    // observer
public:
        virtual ~UsersObserver () { }
        virtual void handle(DatabaseManager* db) {
                // implementation
        };
};
```

```
class NewsObserver : public IObserver {    // observer
public:
        virtual ~NewsObserver () { }
        virtual void handle(DatabaseManager* db) {
                // implementation
        };
};
```



```
class DatabaseManager {
private:
        std::list<IObserver*> m_observers;
        void _notify() {
                for(auto obs : m_observers)
                        obs->handle(*this);
        }
public:
        void add(IObserver* observer) { … }
        void remove(IObserver* observer) { … }
};
```

Observer in real life: Dolphin VCS observer

```
class DOLPHIN_EXPORT VersionControlObserver : public QObject {
    Q_OBJECT
public:
….
    void setModel(KFileItemModel* model);
    KFileItemModel* model() const;
    QList<QAction*> actions(const KFileItemList& items) const;
signals:
    void infoMessage(const QString& msg);
    void errorMessage(const QString& msg);
    void operationCompletedMessage(const QString& msg);
private slots:
….
    void delayedDirectoryVerification();
….
};
```

Observer in real life: Dolphin VCS observer

```
void VersionControlObserver::setModel(KFileItemModel* model) {
    if (m_model) {
        disconnect(m_model, &KFileItemModel::itemsInserted,
                   this, &VersionControlObserver::delayedDirectoryVerification);
        disconnect(m_model, &KFileItemModel::itemsChanged,
                   this, &VersionControlObserver::delayedDirectoryVerification);
    }

    m_model = model;

    if (model) {
        connect(m_model, &KFileItemModel::itemsInserted,
                this, &VersionControlObserver::delayedDirectoryVerification);
        connect(m_model, &KFileItemModel::itemsChanged,
                this, &VersionControlObserver::delayedDirectoryVerification);
    }
}
```

Observer in real life: Dolphin VCS observer

```
// client code

void DolphinView::slotModelChanged(KItemModelBase* current, KItemModelBase* previous) {
    if (previous != nullptr) {
        ....
        m_versionControlObserver->setModel(nullptr);
    }

    if (current) {
         ....
        KFileItemModel* fileItemModel = static_cast<KFileItemModel*>(current);
        ....
        m_versionControlObserver->setModel(fileItemModel);
    }
}
```

Relations

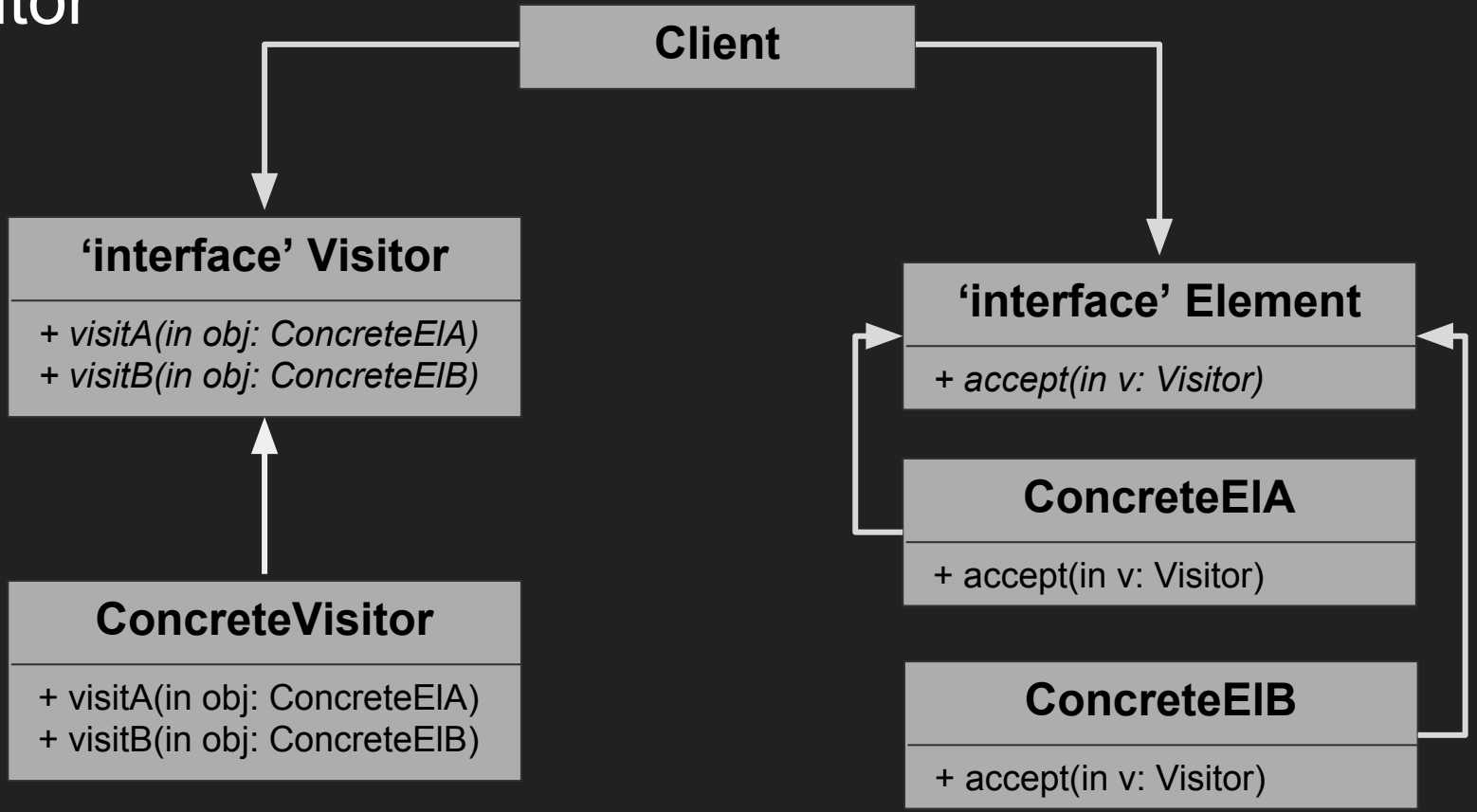
CoR vs Command vs Mediator vs Observer

Когда применять

при изменении состояния одного объекта нужно что-то сделать в другом

одни объекты должны наблюдать за другими, но только в определенных случаях

Visitor


Client
‘interface’ Visitor
+ visitA(in obj: ConcreteElA)
+ visitB(in obj: ConcreteElB)
‘interface’ Element
+ accept(in v: Visitor)
ConcreteElA
+ accept(in v: Visitor)
ConcreteVisitor
+ visitA(in obj: ConcreteElA)
+ visitB(in obj: ConcreteElB)
ConcreteElB
+ accept(in v: Visitor)

Visitor example

```
class IVisitor {
public:
        virtual ~IObserver() {}
        virtual void visit(ClassA* obj) = 0;
        virtual void visit(ClassB* obj) = 0;
};
```



```
class ConcreteVisitor : public IVisitor {
public:
        virtual void visit(ClassA* obj) { … }
        virtual void visit(ClassB* obj) { … }

};
```

. . . . .

```
class IBase {    // element
public:
        virtual ~Base () { }
        virtual void accept(IVisitor* visitor) = 0;
};
```

```
class ClassA : public IBase {    // concrete element
public:
        virtual void accept(IVisitor* visitor) {
                visitor->visit(this);
        };
};
```

```
class ClassB : public IBase {    // concrete element
public:
        virtual void accept(IVisitor* visitor) {
                visitor->visit(this);
        };
};
```

Visitor in real life: KDevelop type system

Классы типов содержат методы для Посетителя

```
void IntegralType::accept0(TypeVisitor *v) const {
        v->visit (this);
}

 void UnsureType::accept0(KDevelop::TypeVisitor* v) const {
                FOREACH_FUNCTION(const IndexedType& type, d_func()->m_types) {
                AbstractType::Ptr t = type.abstractType();
                v->visit(t.data());
        }
}
```

Visitor in real life: KDevelop type system

```
class KDEVPLATFORMLANGUAGE_EXPORT TypeVisitor {
public:
  virtual ~TypeVisitor ();
  virtual bool preVisit (const AbstractType *) = 0;
  virtual void postVisit (const AbstractType *) = 0;
  ///Return whether sub-types should be visited(same for the other visit functions)
  virtual bool visit(const AbstractType*) = 0;

  virtual void visit (const IntegralType *) = 0;

  virtual bool visit (const PointerType *) = 0;
  virtual void endVisit (const PointerType *) = 0;

  virtual bool visit (const ReferenceType *) = 0;
  virtual void endVisit (const ReferenceType *) = 0;

.....
};
```

Relations

Visitor vs Command

Visitor & Composite

Visitor & Iterator

Когда применять

нужно выполнить операцию над всеми элементами сложной структуры

новое поведение имеет смысл только для некоторых классов из существующей иерархии