Builder Pattern

Tomás Felipe Melli

June 24, 2025

Índice

1 Intent **2**

2 Motivation **2**

3 Applicability **2**

4 Structure **2**

5 Participants **2**

6 Example **3**

1 Intent

Separar la construcción de un objeto complejo de su representación, de modo que el mismo proceso de construcción pueda crear diferentes representaciones.

2 Motivation

Se quiere que un lector de documentos RTF pueda convertir textos RTF a múltiples formatos, como ASCII, TeX o widgets editables. Dado que la cantidad de formatos posibles es abierta, es importante poder agregar nuevas conversiones sin modificar el lector.

La solución es desacoplar el lector del formato de salida, configurándolo con un objeto TextConverter que se encargue de la conversión. Mientras el RTFReader analiza el documento, delega la conversión de los tokens a TextConverter.

Cada subclase de TextConverter maneja un formato específico (por ejemplo, ASCIIConverter, TeXConverter, TextWidgetConverter). Estas clases encapsulan la lógica de construcción de la representación final detrás de una interfaz abstracta.

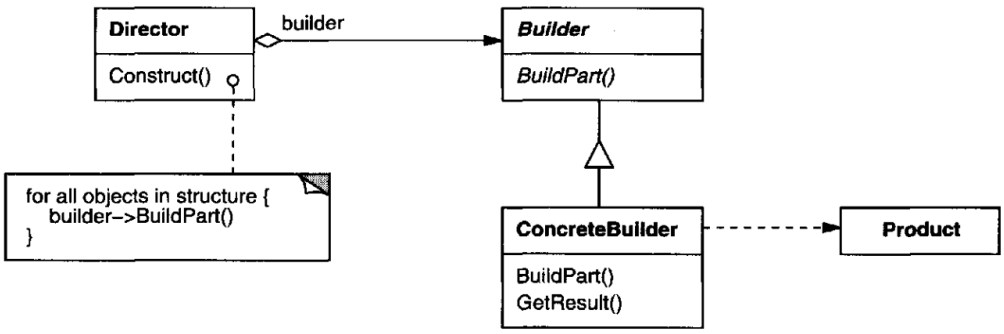
Este diseño aplica el patrón Builder, donde el lector (director) se encarga del análisis, y los convertidores (builders) se encargan de construir distintas representaciones del texto. Así, el mismo proceso de parsing puede generar múltiples salidas configurando el lector con distintos builders.

3 Applicability

Usar Builder cuando:

- El algoritmo para crear un objeto complejo debe ser independiente de las partes que lo componen y de cómo se ensamblan.
- El proceso de construcción debe permitir distintas representaciones del objeto construido.

4 Structure



5 Participants

- Builder (TextConverter): define una interfaz abstracta para crear partes de un objeto complejo (producto).
- ConcreteBuilder (ASCIIConverter, TeXConverter, TextWidgetConverter): implementa la interfaz del builder, construye y ensambla las partes del producto, mantiene su representación interna y ofrece una interfaz para obtener el producto final.
- Director (RTFReader): usa el builder para construir el objeto paso a paso.
- Product (ASCIIText, TeXText, TextWidget): es el objeto complejo que se está construyendo. Sus partes son definidas por el ConcreteBuilder, que controla cómo se ensamblan.

6 Example

Contexto del problema

Interfaz Builder (MazeBuilder)

```
class MazeBuilder {
public:
    virtual void BuildMaze() { }
    virtual void BuildRoom(int room) { }
    virtual void BuildDoor(int roomFrom, int roomTo) { }
    virtual Maze* GetMaze() { return nullptr; }

protected:
    MazeBuilder() = default;
};
```

Director (MazeGame)

```
class MazeGame {
public:
    Maze* CreateMaze(MazeBuilder& builder) {
        builder.BuildMaze();
        builder.BuildRoom(1);
        builder.BuildRoom(2);
        builder.BuildDoor(1, 2);
        return builder.GetMaze();
    }

    Maze* CreateComplexMaze(MazeBuilder& builder) {
        for (int i = 1; i <= 1001; ++i) {
            builder.BuildRoom(i);
        }
        return builder.GetMaze();
    }
};
```

Concrete Builder (StandardMazeBuilder)

```
class StandardMazeBuilder : public MazeBuilder {
public:
    StandardMazeBuilder() : _currentMaze(nullptr) { }

    void BuildMaze() override {
        _currentMaze = new Maze;
    }

    void BuildRoom(int n) override {
        if (!_currentMaze->RoomNo(n)) {
            Room* room = new Room(n);
            _currentMaze->AddRoom(room);
            room->SetSide(North, new Wall);
            room->SetSide(South, new Wall);
            room->SetSide(East,  new Wall);
            room->SetSide(West,  new Wall);
        }
    }

    void BuildDoor(int n1, int n2) override {
        Room* r1 = _currentMaze->RoomNo(n1);
        Room* r2 = _currentMaze->RoomNo(n2);
        Door* d = new Door(r1, r2);
        r1->SetSide(CommonWall(r1, r2), d);
        r2->SetSide(CommonWall(r2, r1), d);
    }

    Maze* GetMaze() override {
        return _currentMaze;
    }

private:
```

```
    Direction CommonWall(Room*, Room*);
    Maze* _currentMaze;
};
```

Concrete Builder Alternativo (CountingMazeBuilder)

Este no lo construye sólo cuenta las habitaciones y puertas.

```
class CountingMazeBuilder : public MazeBuilder {
public:
    CountingMazeBuilder() : _rooms(0), _doors(0) { }

    void BuildRoom(int) override {
        _rooms++;
    }

    void BuildDoor(int, int) override {
        _doors++;
    }

    void GetCounts(int& rooms, int& doors) const {
        rooms = _rooms;
        doors = _doors;
    }

private:
    int _rooms;
    int _doors;
};
```

Resultado

Uso del contador

```
int rooms, doors;
MazeGame game;
CountingMazeBuilder builder;

game.CreateMaze(builder); // realiza el "dibujo" virtual
builder.GetCounts(rooms, doors);

cout << "The maze has " << rooms << " rooms and "
     << doors << " doors" << endl;
```

Uso del patrón completo

```
MazeGame game;
StandardMazeBuilder builder;

game.CreateMaze(builder);
Maze* maze = builder.GetMaze();
```