Builder Pattern – Creational

- Intent

  Separate the construction of a complex object from its representation so that the same construction process can create different representations

- Motivation

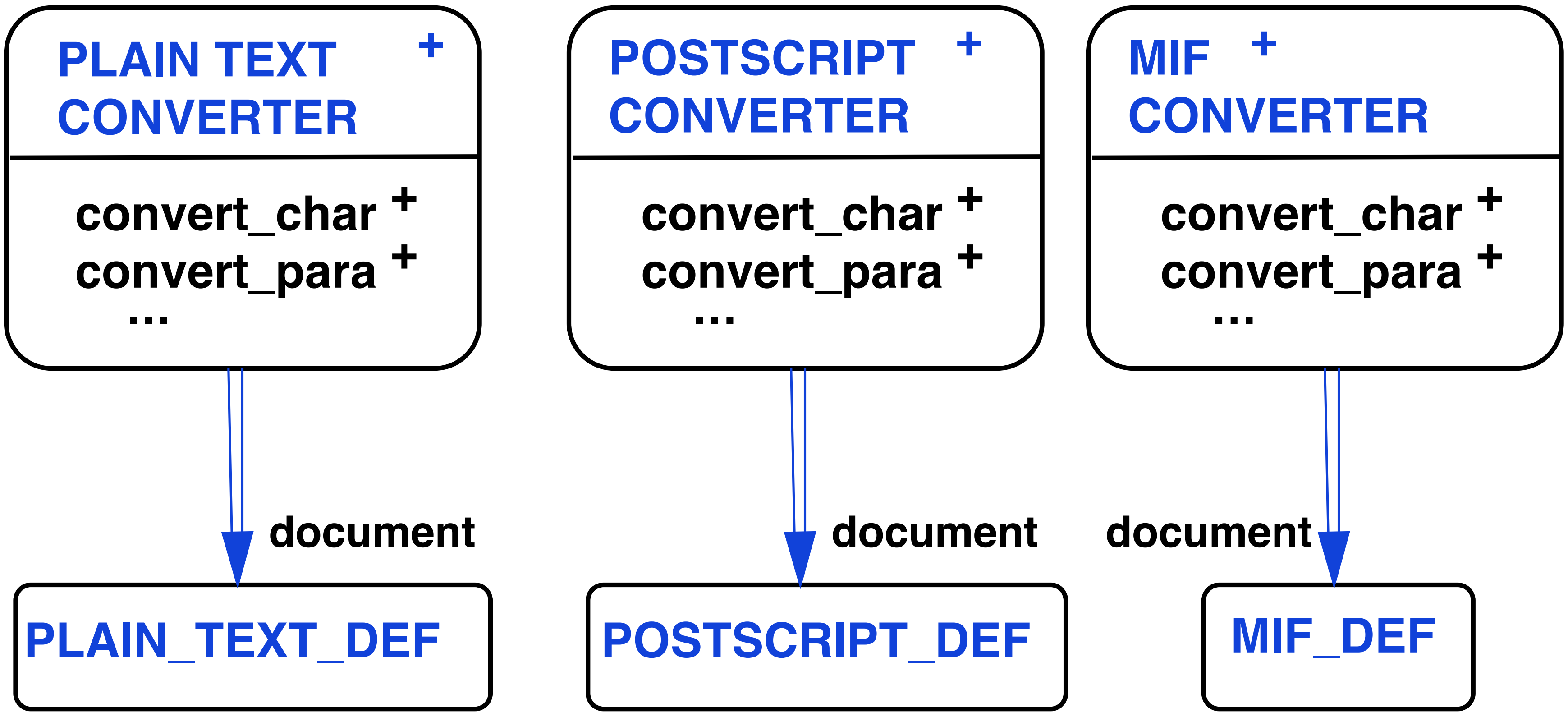
  » **Reader for RTF (Rich Text Format) should be able to convert to any other representation**

    Plain Text, MIF (Maker Interchange File), Postscript

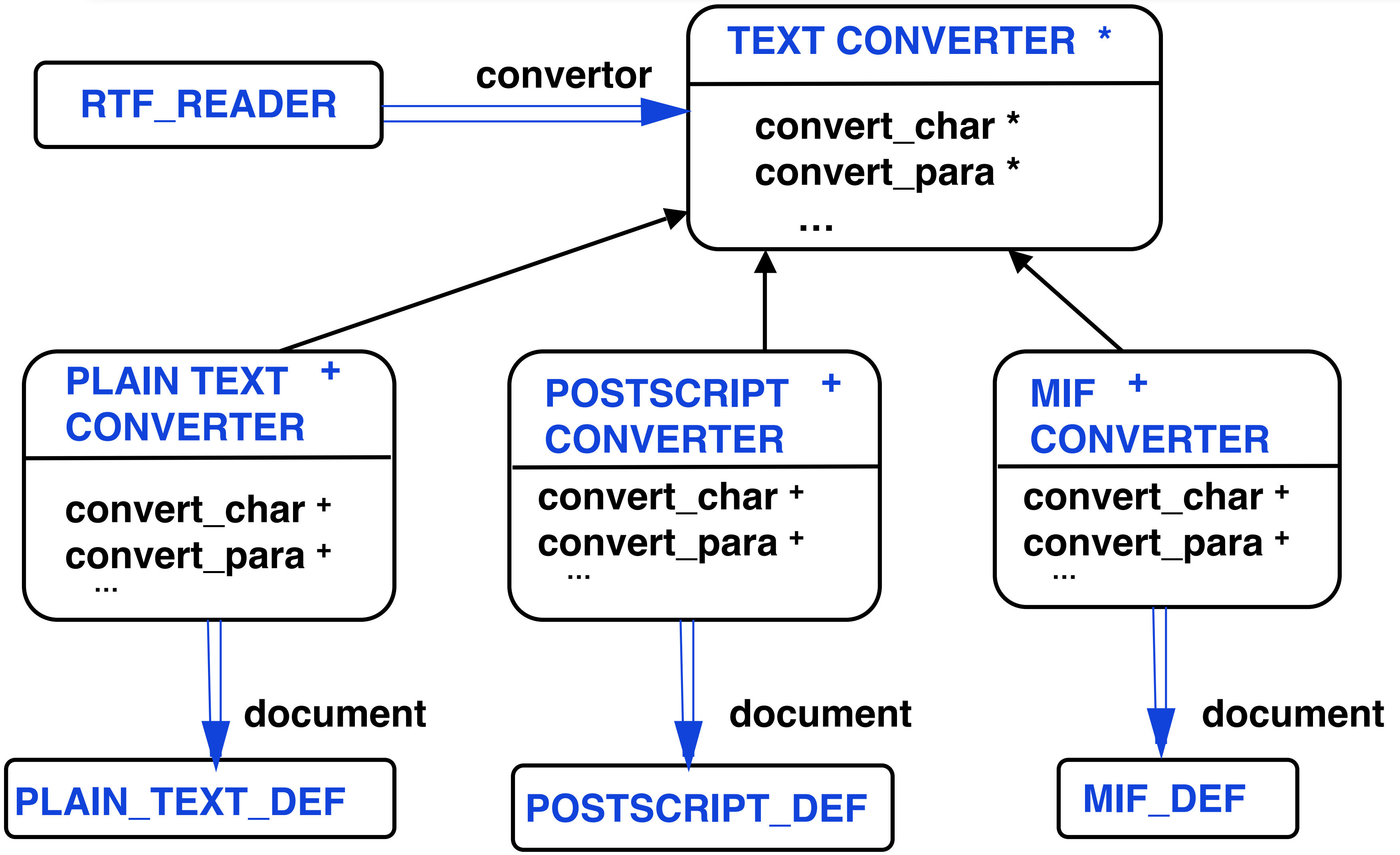
  » **Open ended number of representations possible**

  » **Abstract the conversion process**

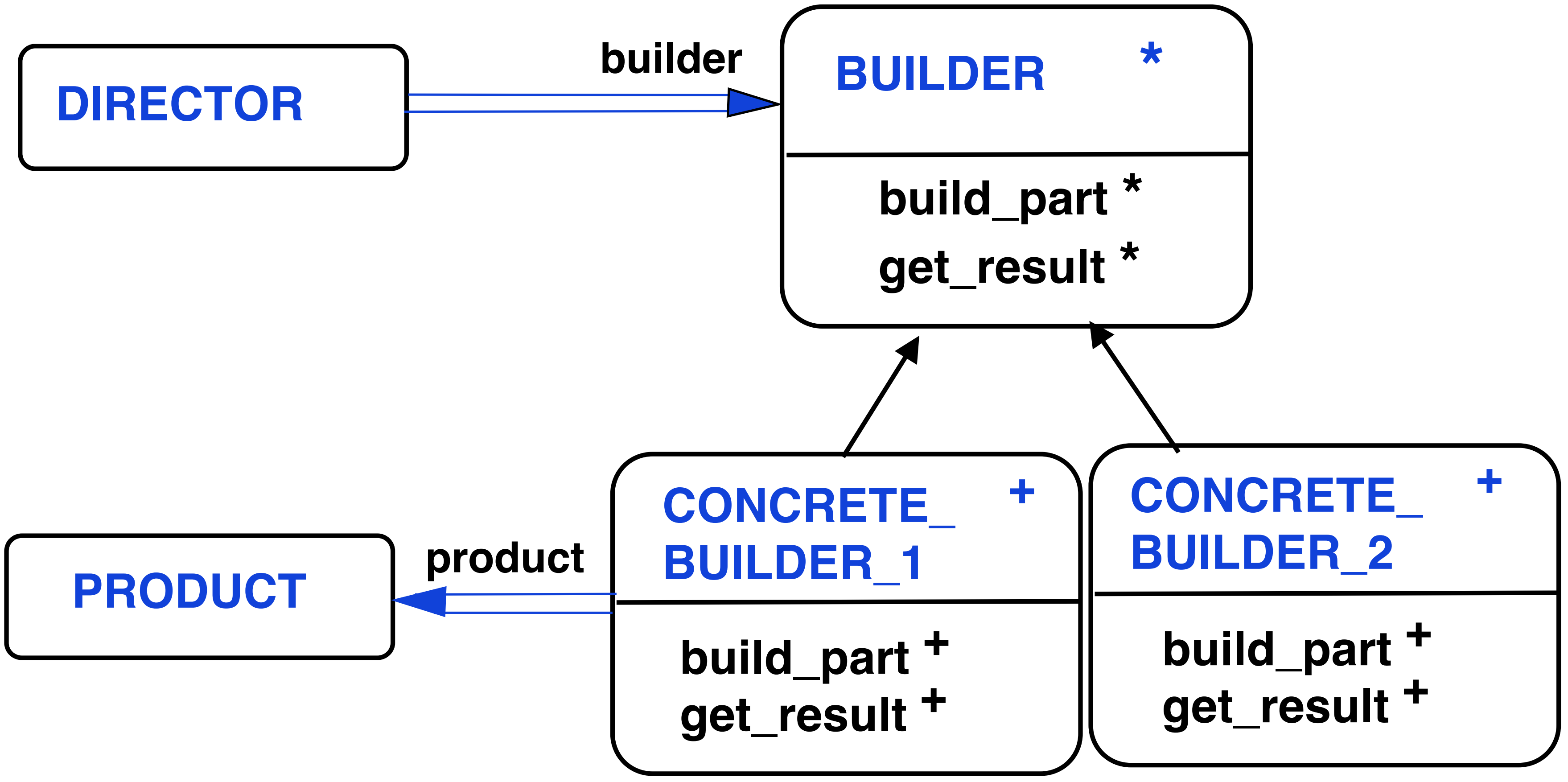
Example Conversions



Example Architecture



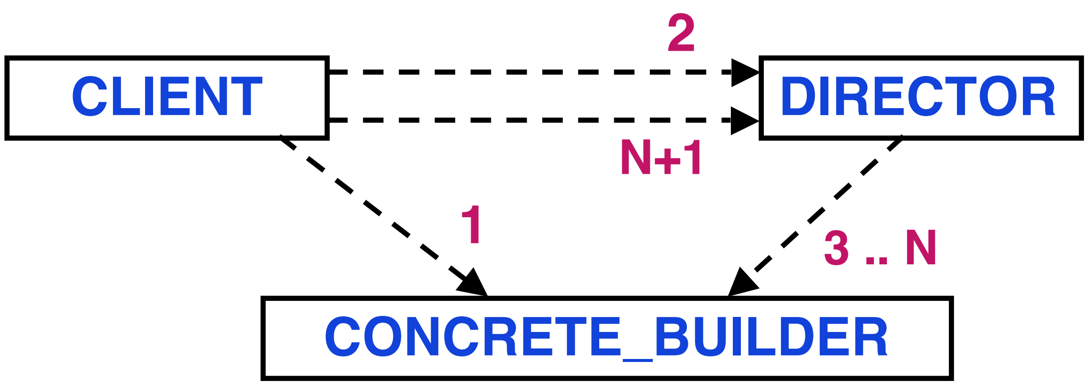
Abstract Architecture



Scenario

Scenario: Build a product

1 create theBuilder.make
2 director.make_with(theBuilder)
3 theBuilder.build_part_1
4 theBuilder.build_part_2
...
N theBuilder.build_part_N
N+1 director.get_product



Participants

- Builder

  Specifies abstract interface for creating parts of a product object

- Concrete builder

  Constructs and assembles parts of the product by implementing the Builder interface

- Director

  Constructs an object using the Builder interface

- Product

  » **The complex object under construction**

  » **Includes classes that define the parts and interfaces for assembling parts into a final result**

Applicability

- When algorithm for creating a complex object should be independent of the parts that make up the object and how they are assembled

- The construction process must allow different representations for the object that is constructed

Collaboration

- The client creates a Director object and configures it with the desired Builder object
- Director notifies Builder whenever a part of the product should be built
- Builder handles requests from the Director and adds parts to the product
- Client retrieves the product from the Builder

Builder Definition

```
deferred class BUILDER
feature

  product : ANY          -- Return the built product
      deferred end


   build             -- Build the complete product
     deferred
     ensure
           product_not_void:  product /= void
     end

end
```

Maze Builder

```
deferred class MAZE_BUILDER
   inherit BUILDER
               rename product as maze, build as build_maze
               redefine maze  end
feature
   maze: MAZE -- The maze being built
         deferred end

   build_maze  -- Build a complete maze
         deferred end
   build_room ( room_id : STRING )
         -- Build a single room
         deferred end

   build_door ( room_1_id, room_2_id : STRING )
         -- Put a door between the identified rooms
         deferred end
```

Basic Maze Builder

```
class BASIC_MAZE BUILDER
        inherit BUILDER
feature
  maze: MAZE              -- The maze being built

  build_maze              -- Build a complete maze
        do create maze.make end


  build_room ( room_id : STRING )
        -- Build a single room

        do … custom implementation … end

  build_door ( room_1_id, room_2_id : STRING )
        -- Put a door between the identified rooms
        do … custom implementation … end

end
```

Enchanted Maze Builder

```
class ENCHANTED_MAZE BUILDER
          inherit BUILDER
feature
   maze: MAZE              -- The maze being built

   build_maze              -- Build a complete maze
          do create maze.make end


   build_room ( room_id : STRING )
          -- Build a single room

          do … custom implementation … end

   build_door ( room_1_id, room_2_id : STRING )
          -- Put a door between the identified rooms
          do … custom implementation … end

end
```

Bombed Maze Builder

```
class BOMBED_MAZE BUILDER
        inherit BUILDER
feature
  maze: MAZE              -- The maze being built

  build_maze              -- Build a complete maze
        do create maze.make end


  build_room ( room_id : STRING )
        -- Build a single room

        do … custom implementation … end

  build_door ( room_1_id, room_2_id : STRING )
        -- Put a door between the identified rooms
        do … custom implementation … end

end
```

Common Build

```
deferred class COMMON_BUILD
feature
    maze : MAZE
    make  deferred  end
    create_maze ( builder : MAZE_BUILDER ) : MAZE
    local r1_id, r2_id : STRING
    do
        builder.buld_maze
        r1_id := "Room 1"    ;    r2_id := "Room 2"
        builder.build_room (r1_id)  ;  builder.build_room (r2_id)
        builder.build_door (r1_id , r2_id)
        Result := builder.maze
    end
```

Basic Builder

```
class BASIC_BUILDER
         inherit COMMON_BUILD

create make

feature
   make
      local maze_builder : MAZE_BUILDER

      do
         create { BASIC_MAZE_BUILDER } maze_builder
         maze := create_maze ( maze_builder )
      end
```

ENCHANTED Builder

```
class BASIC_BUILDER
          inherit COMMON_BUILD

create make

feature
   make
      local maze_builder : MAZE_BUILDER

      do
         create { ENCHANTED_MAZE_BUILDER } maze_builder
         maze := create_maze ( maze_builder )
      end
```

Bombed Builder

```
class BASIC_BUILDER
          inherit COMMON_BUILD

create make

feature
   make
      local maze_builder : MAZE_BUILDER

      do
         create { BOMBED_MAZE_BUILDER } maze_builder
         maze := create_maze ( maze_builder )
      end
```

Builder Client

```
make
    local
            maze_1: BASIC_BUILDER
            maze_2: ENCHANTED_BUILDER
            maze_3: BOMBED_BUILDER
    do
            create maze_1 . make
            maze_1 . describe

            create maze_1 . make
            maze_1 . describe

            create maze_1 . make
            maze_1 . describe
    end
```

Related Patterns

- Abstract Factory focuses on families of product objects, while Builder focuses on step by step construction of complex objects

- Builder frequently builds a Composite