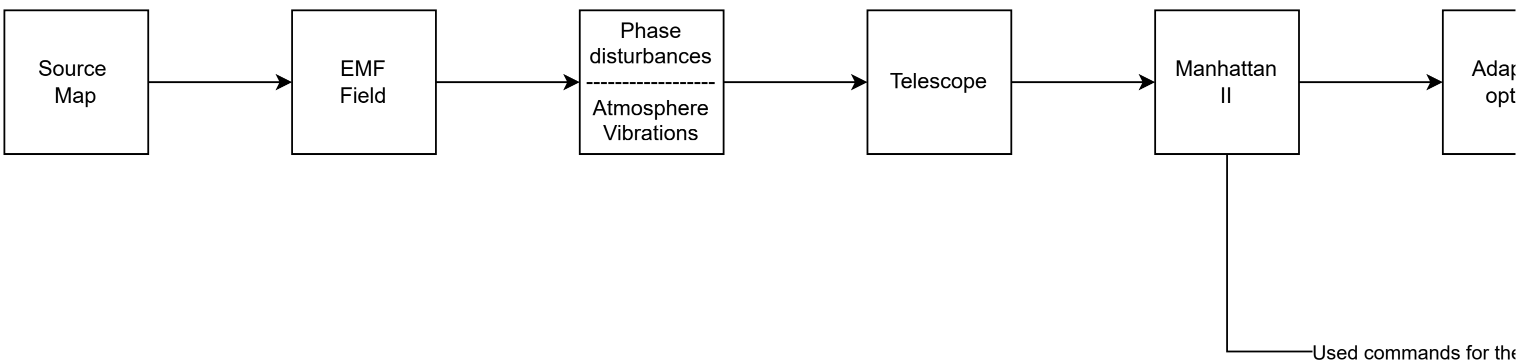


Motivation of the structure – top-down requirements:

- The system is built from class objects that modify a major class; the Field class – class contains the EMF of light as it propagates from object
- The field is modified by the Objects to reflect the underlying physical process – e.g., the atmosphere object receives a Field class and adds th
- To simplify the code structure, the object needs to be de-coupled. The input of a code must be only the Field object, and the modification shou
- Each of the object classes is built from an abstract class that must include the following:
  - **Initialize function** – initializes the object and loads the parameters from the preamble section.
  - **Propagation function** – the function that picks up the field from the previous object and operates the field to reflect the physical proce baselines x 4 ABCD inputs).
  - **The state function** yields the current state of the object – e.g., this will include the mirror commands in an AO class; the delay line pos

Structure – basic code idea:

Preamble – parameter loading. Reads parameter files using a reader function

1. Telescope/atmospheric/source parameters; AO/FT/MAH2 configuration; Beam combiner matrices; source parameters; DL/Detector/Spectrom

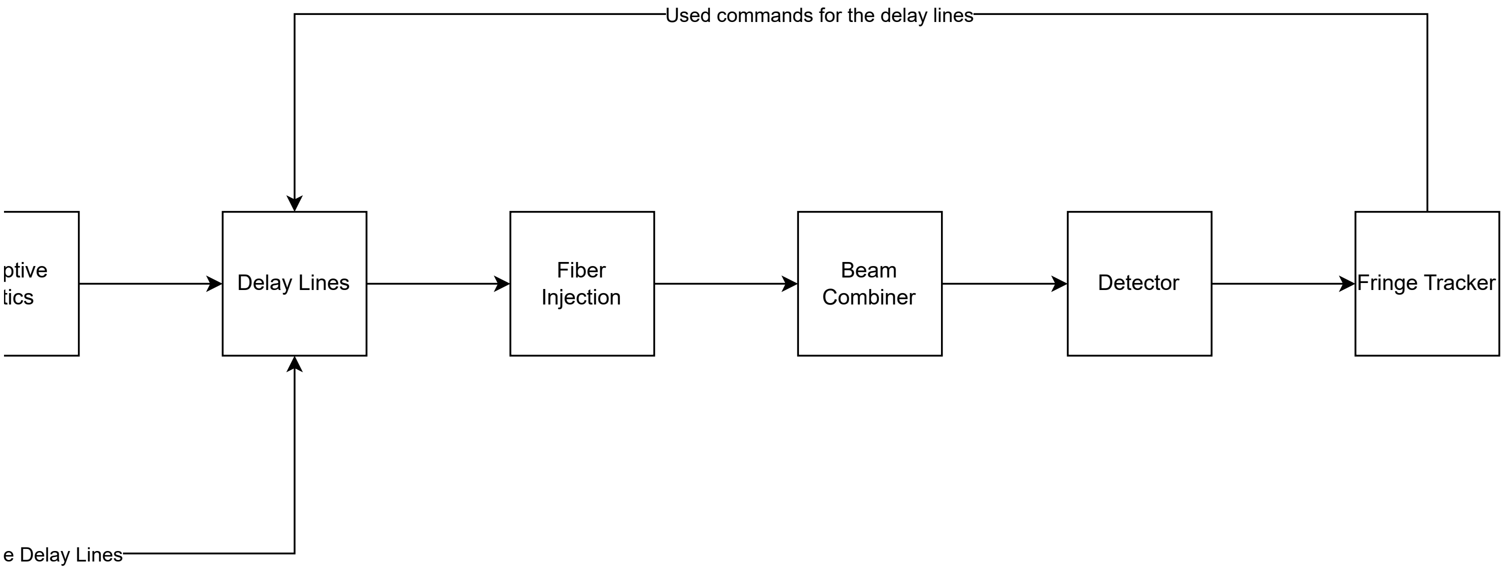
Initialization of the field - The field object is generated from the Source Object - it is never generated independently in a VLTI loop.

1. source = source.initialize(source parameters);
2. emf = source.generate_field()

Propagation of the field across the other Digital Twin components:

1. atmosphere = atmosphere.initialize(atmospheric parameters);
2. emf = atmosphere.propagate(emf);

Note on control strategies: The control strategies applied to each control element are fixed for a loop, which implies that the current strategy (e.g., F



: to object.
he turbulent phase to the EMF.
uld operate only on the Field object.

ess of the current object – e.g., a beam combiner class splits the introduced four telescopes into 24 outputs in an ABCD configuration (6

sitions in the Delay lines class.

eter static parameters; …

FT using an integrator) needs to be included in the initialization function and loaded from the parameters.

Structure – basic pseudo-code loop:

```
## Preamble

objects = [Atmosphere, Phase_Disturbances, Telescope, Manhattan, Adaptive_Optics, Delay_Lines, Fiber_injection, Beam_combine

param_objects = load_parameters( parameter_file.txt )

# Load parameters into objects

Source.initialize(source_parameters)
[current_object.initialize(current_parameters) for current_object, current_parameters in zip(objects, param_objects) ]


# Loop

initial_emf = Source.generate_field()

for frame in range( number_of_iterations):

    current_emf = initial_emf

    for current_object in objects:

        current_emf = current_object.propagate(current_emf)

        #Analysis functions if needed:

        ## Example:

        if current_object == Adaptive_Opics:

            current_mirror_commands = ao.state().mirror_commands()

            var_cmds = np.var(current_mirror_commands, axis=1)
```

Note: Tiago comenta que podemos em vez de dar update do emf podemos gerar um novo objecto. Gasta mais memoria, mas pode ser util. Em vez
Note: P.P.G pede que o sistema seja capaz de ingerir scintilação, mesmo que esta nao seja incluida para já.

er, Detector, Fringe_tracker]

eu uso uma state function emf.state() para chamar o estado do campo caso seja necessário.