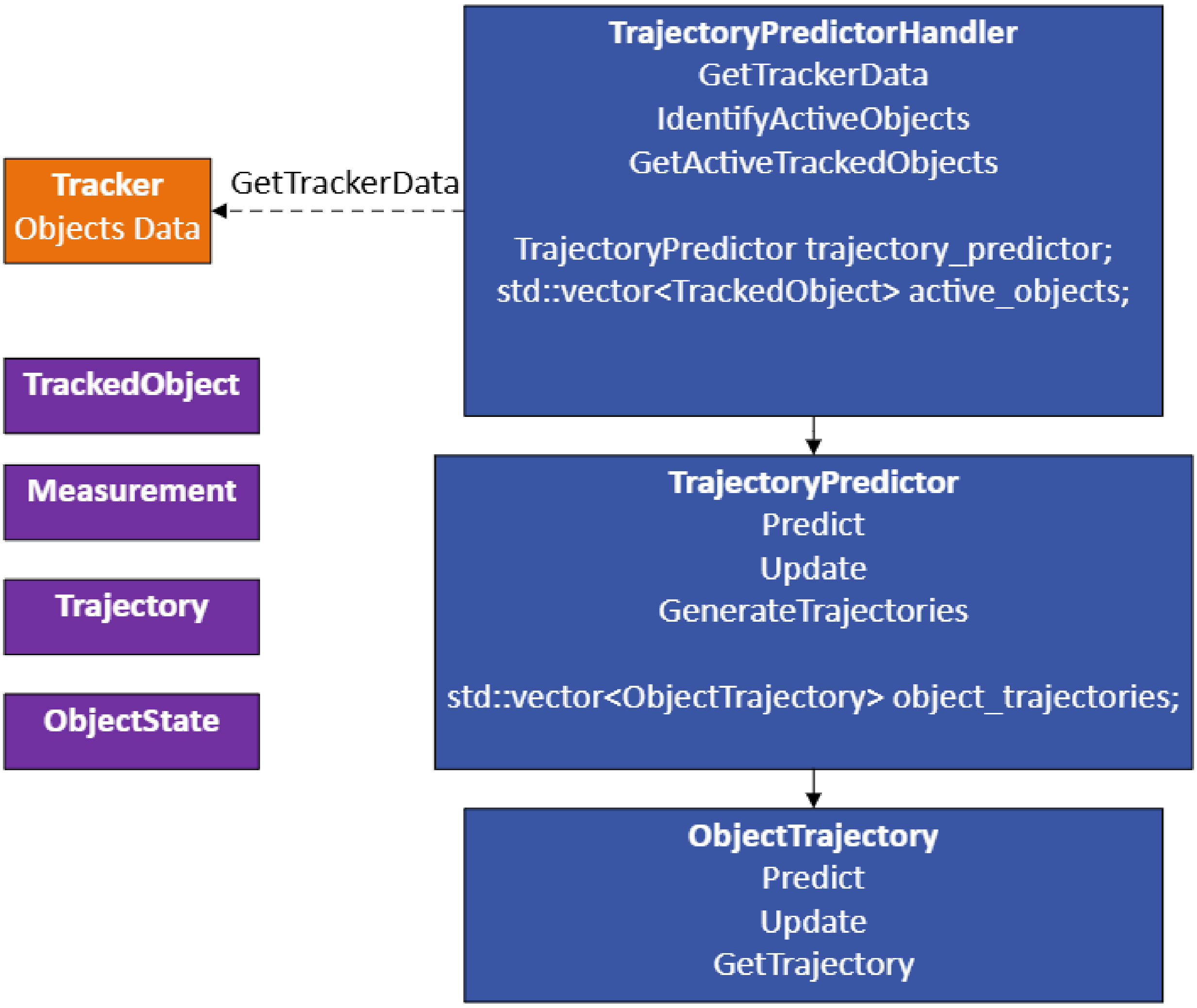
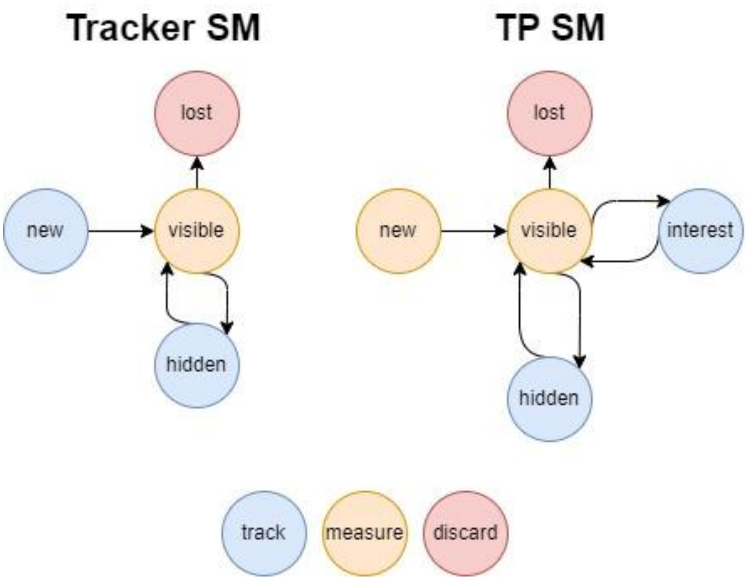
Trajectory Prediction

API



Object State Machine



API

TrajectoryPredictor.h

(You can open above file in VS so it's more readable)

```
class TrajectoryPredictorHandler {
public:
    //Constructor initializes trajectory_predictor
    TrajectoryPredictorHandler(TrajectoryPredictor trajectory_predictor);

    //Parses tracker input into active_objects
    void GetTrackerData();

    //assigns active_objects from filtered (active) objects read by tracker
    //this might be implemented in GetTrackerData instead
    //or perhaps something more complicated using a state machine so, so the filtering of
active objects isn't done at every time step
    void IdentifyActiveObjects();

    //Uses trajectory_predictor to generate trajectories for active objects.
    //Calls GenerateTrajectories from TrajectoryPredictor
    std::vector<ObjectTrajectory> GetActiveTrackedObjects();

private:
    TrajectoryPredictor trajectory_predictor;
    std::vector<TrackedObject> active_objects;
};

//Stores and is responsible for calculating trajectories of all relevant objects using its
predict and update functions.
//May use single objects' predict and update functions for this purpose, as well as some
pre/post processing ontop of these functions
//i.e. perform collision avoidance post-processing like in the interaction-aware IMM
filter approach
class TrajectoryPredictor {
public:
    //calls objects' Predict()
    void Predict();

    //calls objects' Update()
    void Update(Measurement m);

    //Given a list of active objects (with new measurements inside), calculate a new set
of trajectories
    //Calls UpdateActiveObjects from below to update the object_trajectories vector
(add/remove/keep objects)
    void GenerateTrajectories(std::vector<TrackedObject> active_objects);

    std::vector<ObjectTrajectory> GetObjectTrajectories(){ return object_trajectories};
private:
    //Updates the object_trajectories vector so that it reflects the current set of
relevant objects
    //May delete a ObjectTrajectory, or create and initialize a new one.
    //After call, every object in object_trajectories can be called with update() and
predict() with new measurements
    void UpdateActiveObjects(std::vector<TrackedObject> active_objects);
    std::vector<ObjectTrajectory> object_trajectories;
};

class ObjectTrajectory {
public:
    //updates current trajectory with next-step prediction
    void Predict();

    //updates state and trajectory in accordance to newe measurement
    void Update(Measurement m);

    Trajectory GetTrajectory(){ return trajectory };
private:
    Trajectory trajectory;
};

//Tracked Object
struct {
    size_t id;
    Measurement current_measurement;
    ObjectState state;
} TrackedObject;

enum class ObjectState {
    kNew,         // low confidence
    kVisible,     // stable
    kInvisible,   // phantom
    kLost,        // to be terminated
};

//Trajectory base class
struct {
    //trajectory descriptors.
    //Might be implementation-dependant.
    //May contain waypoints, probabilities, sampling generation function, ...
} Trajectory;

//Measurement base class
struct {
    //probably just x,y.
    //Possibly more, since we can directly take tracker's speed, heading measurements.
Possibly implementation-dependant
} Measurement;
```