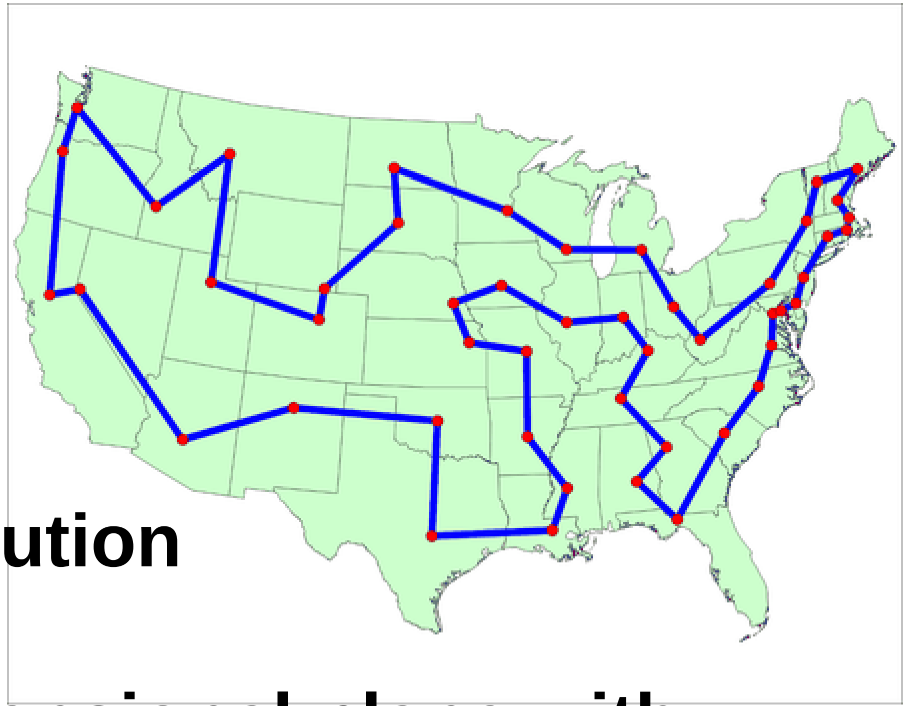
Unmanned Systems Autonomy Services: Task Assignment Pipeline

Dr. Derek Kingston
Control Science Center of Excellence
Aerospace Systems Directorate
Air Force Research Laboratory

Integrity ★ Service ★ Excellence

Traveling Salesman Problem



- Problem: an agent must visit all cities in a tour with minimal travel cost
- Simple to describe, extremely difficult to reach precise optimal solution
- Classic NP-hard problem
- For "cities" on a two-dimensional plane with symmetric travel costs, very powerful heuristics can solve large problems to near-optimally

Generalized Task Assignment

- Consider N agents and M tasks
- Costs are incurred for reaching and completing tasks

- Problem: allocate M tasks across N agents such that all tasks are completed and the sum of all costs incurred is minimized

- Note, when N=1, this is a version of the Traveling Salesman Problem

Application to Cooperative Control

- Interesting multi-vehicle missions are often variations on the generalized task assignment problem
- Variations:
  - Boolean constraints
  - Temporal constraints
  - Asymmetric (often from dynamic constraints)
  - Heterogeneous agents
  - Multi-objective/multi-cost
- Typically overwhelmed with choices; different techniques needed for highly constrained spaces

Application to Cooperative Control

- Interesting multi-vehicle missions are often variations on the generalized task assignment problem
- Variations:
  - Boolean constraints
  - Temporal constraints
  - Asymmetric (often from dynamic constraints)
  - Heterogeneous agents
  - Multi-objective/multi-cost
- Typically overwhelmed with choices; different techniques needed for highly constrained spaces

AFRL

Task-Centric View

- **Key design decision: task representation**
- **UxAS uses a generic task model: all tasks connect to the rest of the pipeline in the exact same way**
- **Each task offers up a series of “options”, i.e. different ways that a task can be carried out *by a single vehicle***
- **Mathematically, each option consists of a *start* pose, an *end* pose, and a cost to complete**
- **A “task” is then:**
  - **A set of options (and agent eligibility)**
  - **The relationship between those options**

TaskOption {uxas.messages.task.TaskOption} ID = 19

field of: TaskPlanOptions,

Start/end locations and headings and cost for implementing the task from this configuration

Field	Type	Units
TaskID *Task ID* *Default Value = 0*	int64	None
OptionID *ID for this option* *Default Value = 0*	int64	None
EligibleEntities *Eligible entities for completing this option with identical cost to complete. If list is empty, then all vehicles are assumed to be eligible.* *Default Value = 0*	int64[]	None
Cost *Cost to complete option in terms of time (given in milliseconds)* *Default Value = 0*	int64	milliseconds
StartLocation *Start location entering the option*	Location3D	None
StartHeading *Start heading entering the option* *Default Value = 0*	real32	degrees
EndLocation *Ending location for this option*	Location3D	None
EndHeading *Ending heading for this option* *Default Value = 0*	real32	degrees

Task Option Reporting

- **When an automation request is made, each task is responsible for reporting its "options"**

TaskPlanOptions {uxas.messages.task.TaskPlanOptions}		ID = 20
Summary of available options to complete this task		
Field	**Type**	**Units**
CorrespondingAutomationRequestID *ID that matches this message with the appropriate unique automation request* *Default Value = 0*	int64	None
TaskID *Task ID* *Default Value = 0*	int64	None
Composition *Process algebra string encoding all of the different options*	string	None
Options *List of options*	TaskOption[]	None

Example

- A “line search task” is described by
  - An ordered list of points
  - A desired view angle
- Only two options produced
  - p1: Search along the line in the order defined
  - p2: Search along the line in the reverse order defined
- The Process Algebra Composition String is then:
  - +(p1 p2)
- During execution
  - Uses “Gimbal Angle Action” message to steer camera

Process Algebra

- Process Algebra is the “glue” that connects all the disparate options to tasks and to missions
- Three main relationships:
  - OR, choice point, represented as +
  - AND, sequential operator, represented as .
  - PARALLEL, any order, represented as |
- All Process Algebra fields in UxAS are strings and are connections of propositions (options) with *prefix* notation
  - Each option is represented as ‘p{optionID}’
  - Prefix notation with operator preceding operands in parentheses

Process Algebra: Examples

- Area search that can be decomposed into 2 sub-regions
  - p1: entire region
  - p2: western half
  - p3: eastern half
  - p4: northern half
  - p5: southern half
- Any vehicle with appropriate sensor that meets resolution requirements is eligible for all options
- Process Algebra relationship is then:
  +(p1 |(p2 p3) |(p4 p5))

Process Algebra: Examples

- Three vehicle cordon (block intersections)
  - p11: vehicle 1 blocks intersection A
  - p12: vehicle 1 blocks intersection B
  - p13: vehicle 1 blocks intersection C
  - p21: vehicle 2 blocks intersection A
  - p22: vehicle 2 blocks intersection B
  - p23: vehicle 2 blocks intersection C
  - etc
- Process Algebra relationship:

+(|(p11 p22 p33) |(p11 p23 p32)
|(p12 p21 p33) |(p12 p23 p31)
|(p13 p21 p32) |(p13 p22 p31))

Multi-Vehicle Tasks

- The number of options and the complexity of the Process Algebra relationship can explode
  - Using options for precise vehicles and then encoding team permutations is a bit of an abuse of the Process Algebra intent
  - If pre-optimization can be used to heuristically determine only "possibly good" choices, then can dramatically reduce complexity growth
- Process Algebra has no capacity for specifying time – how can the multiple vehicles be ensured to be on-task simultaneously?
  - A: pre-pend a "rendezvous" task (in progress)

Task Assignment

Phase 1 – Calculate visible paths and costs

Phase 2 – Calculate preliminary costs for task option
- Calculate lower cost bounds for moving each vehicle to each task from each task to every other task

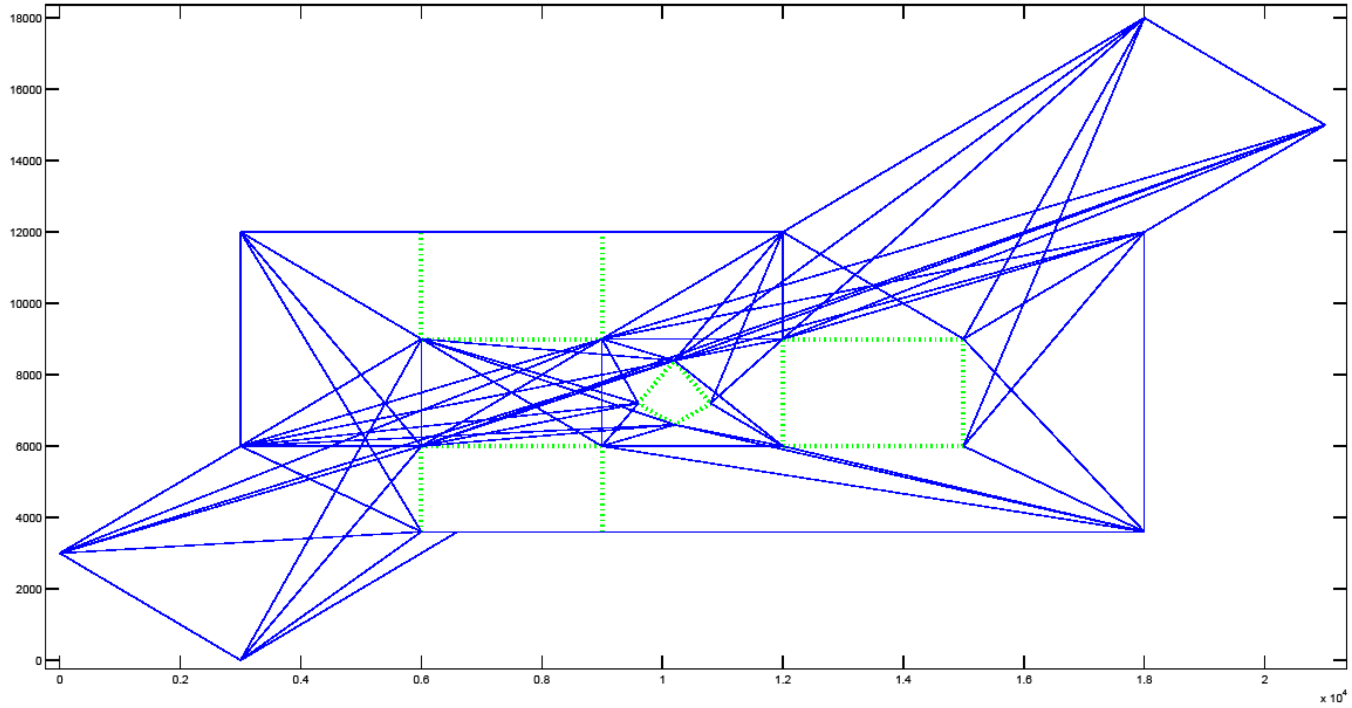
Phase 3 – Optimize cost of assigning vehicles to tasks

Phase 4 – Post-process assignments into waypoints

AFRL

Task Assignment

- **Phase 1: Generate Baseline Paths**
  - **Inputs**: persistent (or slow-changing) information, such as no-fly zones, terrain
  - **Outputs**: Visibility graph creation

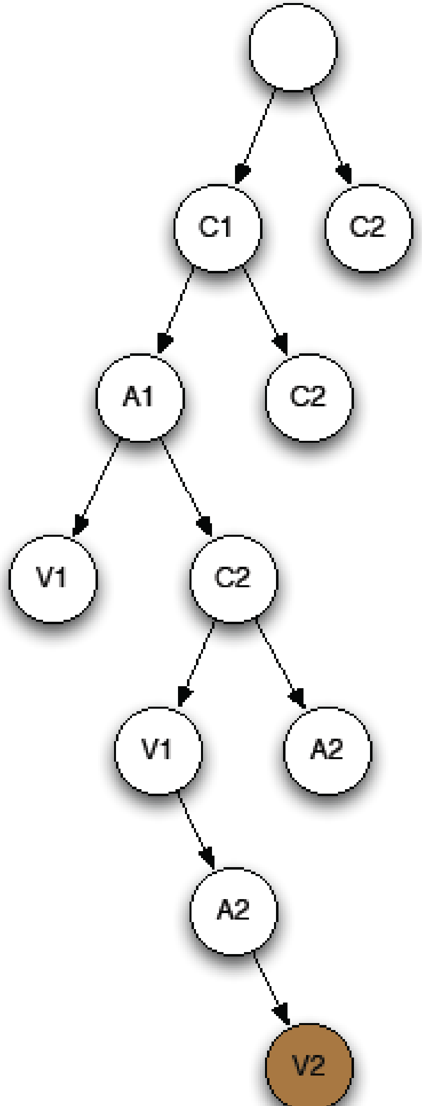


Task Assignment

- **Phase 2: Task Route Generation and Inclusion**
  - **Inputs: Tasks with associated parameters, vehicle initial positions**
  - **Computation: Call task specific route generation; build list of options for each task (direction, method choice); merge start/end positions and vehicle positions into visibility graph**
  - **Outputs: Lower bound on cost-to-go between all tasks and vehicle positions; stored lists of camera actions during each task**

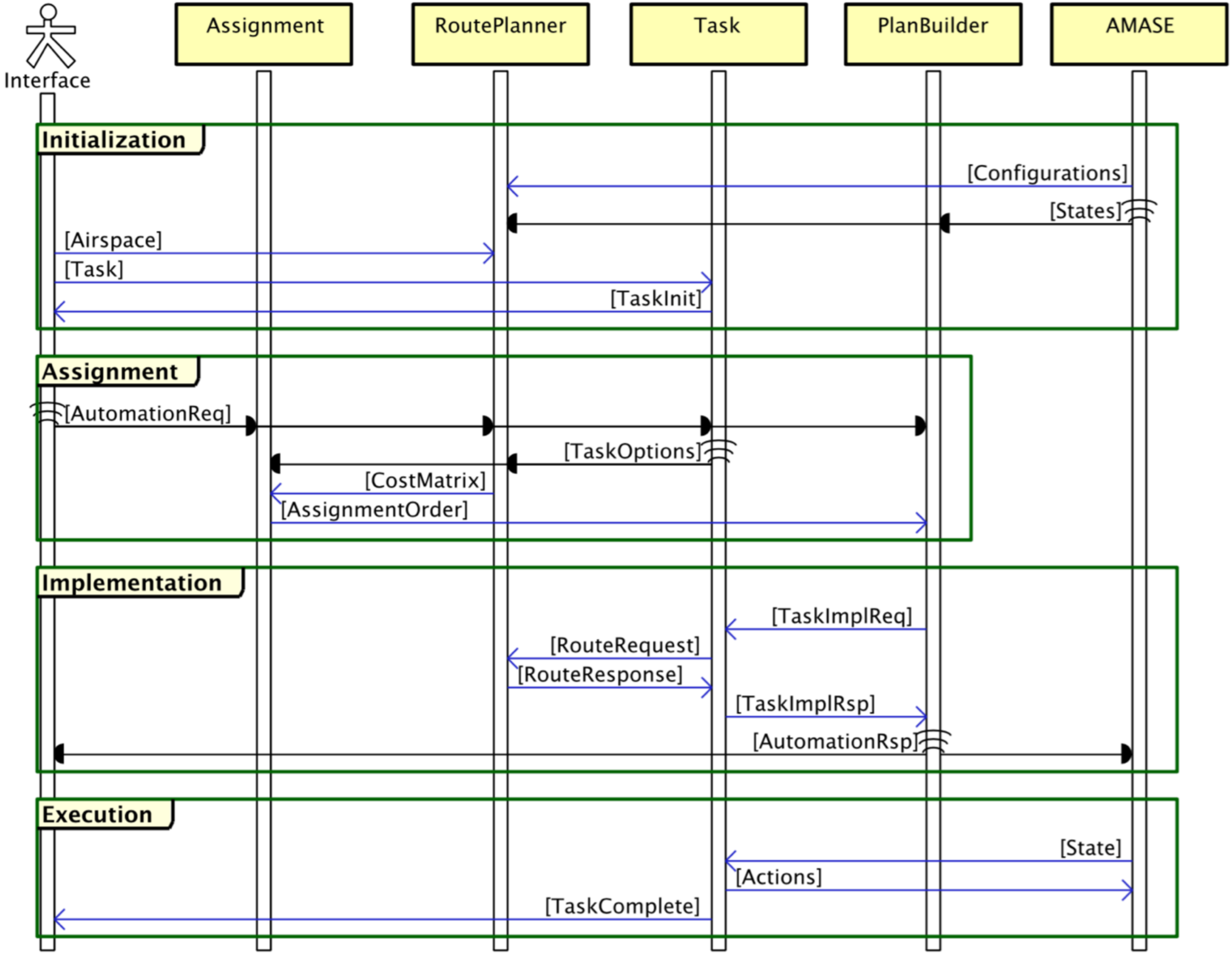
Task Assignment

- **Phase 3: Tree Search**
  - **Inputs**: Lower-bound cost-to-go for each (task,task) and (task,vehicle) pair; algebraic relationship between tasks (AND/OR/Precedence relationships); upper bound on computation time
  - **Computation**: best-first search; improvement over time via branch-and bound
  - **Outputs**: Task order and assignment for each vehicle

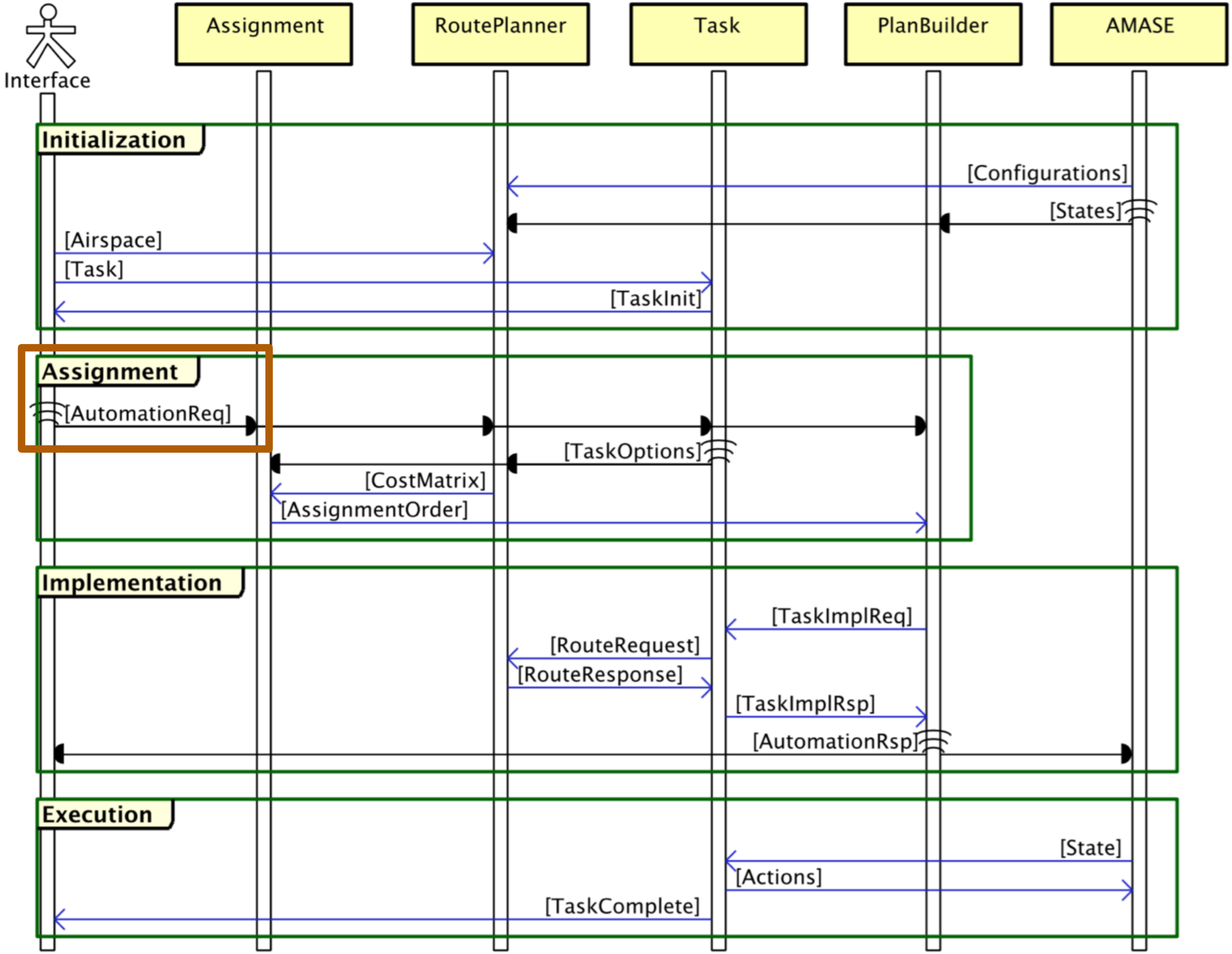


Task Assignment

- **Phase 4: Trajectory Generation and Implementation**
  - **Inputs**: Task ordering; lower-bound paths; current vehicle position
  - **Computation**: Connects tasks with feasible, smoothed trajectories; uploads waypoints
  - **Outputs**: Camera actions during task execution

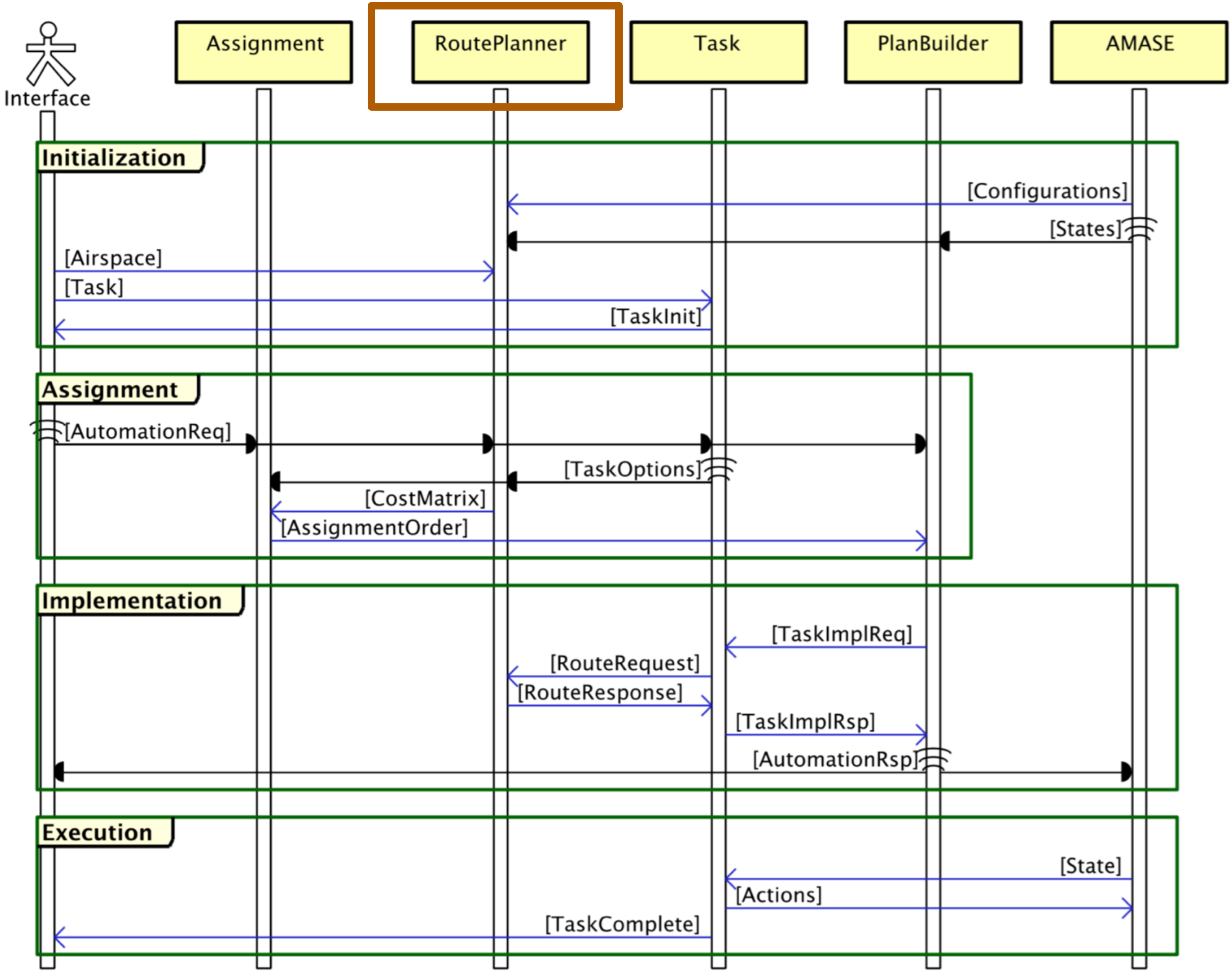


Interface
Assignment
RoutePlanner
Task
PlanBuilder
AMASE
Initialization
[Configurations]
[States]
[Airspace]
[Task]
[TaskInit]
Assignment
[AutomationReq]
[TaskOptions]
[CostMatrix]
[AssignmentOrder]
Implementation
[TaskImplReq]
[RouteRequest]
[RouteResponse]
[TaskImplRsp]
[AutomationRsp]
Execution
[State]
[Actions]
[TaskComplete]


Interface
Assignment
RoutePlanner
Task
PlanBuilder
AMASE
Initialization
[Configurations]
[States]
[Airspace]
[Task]
[TaskInit]
Assignment
[AutomationReq]
[TaskOptions]
[CostMatrix]
[AssignmentOrder]
Implementation
[TaskImplReq]
[RouteRequest]
[RouteResponse]
[TaskImplRsp]
[AutomationRsp]
Execution
[State]
[Actions]
[TaskComplete]

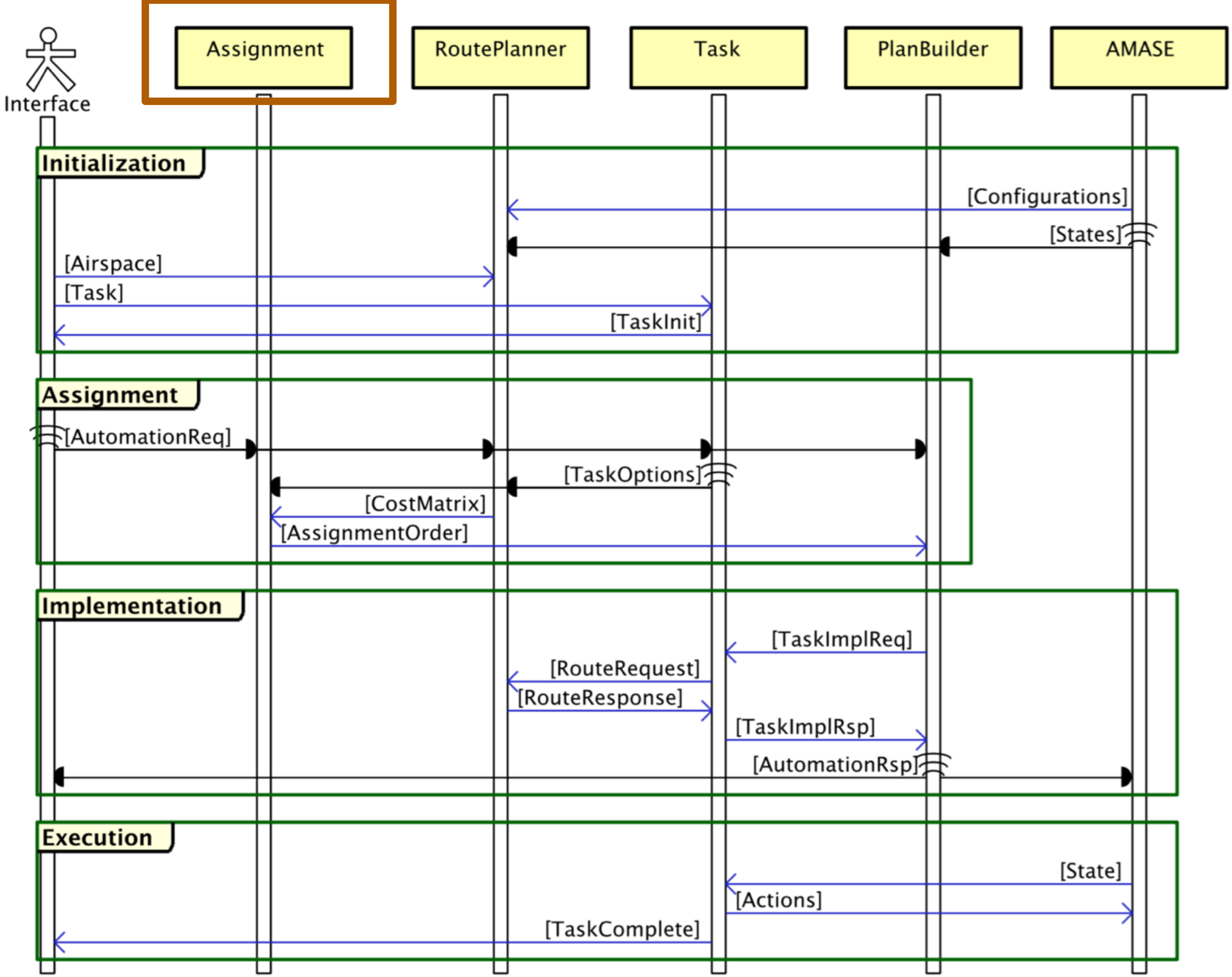
Starting the Pipeline

- Works in "single shot" manner:
  - A single request is completely processed before con't
  - New requests override previous requests
    - Implication: if a vehicle was part way through a task and then included in a new request, its mission has been redefined and will abandon previously assigned tasks
  - Maintaining continuity between automation requests requires carefully designed higher-level management
- Key message: Automation Request
  - Vehicles to consider
  - Tasks to complete
  - Operating Region to respect


Interface
Assignment
RoutePlanner
Task
PlanBuilder
AMASE
Initialization
[Configurations]
[States]
[Airspace]
[Task]
[TaskInit]
Assignment
[AutomationReq]
[TaskOptions]
[CostMatrix]
[AssignmentOrder]
Implementation
[TaskImplReq]
[RouteRequest]
[RouteResponse]
[TaskImplRsp]
[AutomationRsp]
Execution
[State]
[Actions]
[TaskComplete]

Route Planner

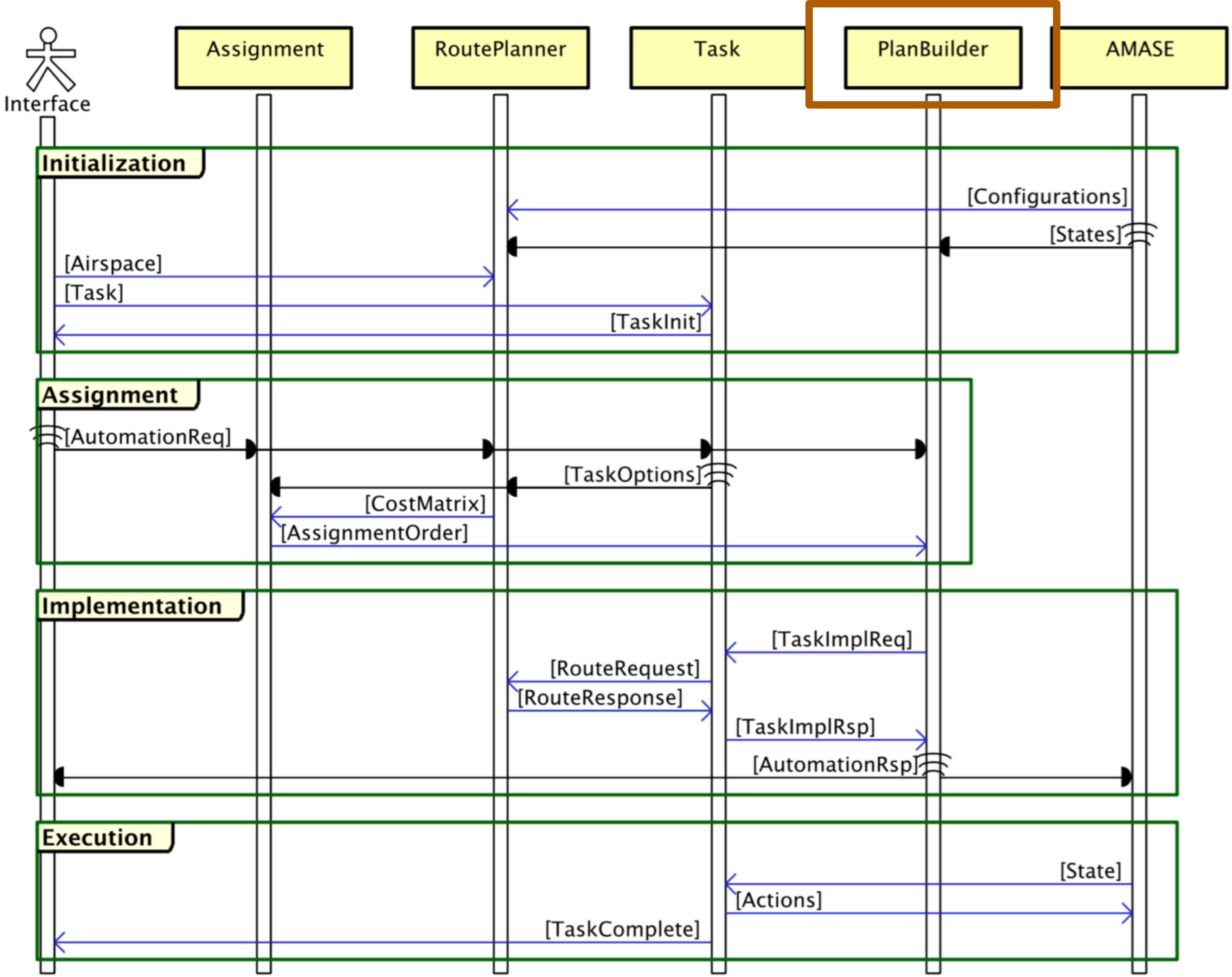
- Service that provides route planning using a visibility heuristic
- Fundamental architectural decision in UxAS: separation of route planning from task assignment
- Intended to be as simple as possible: a route planning service considers routes only in fixed environments for known vehicles and handles requests for single vehicles
- Necessary background data messages:
  - Keep In Zone
  - Keep Out Zone
  - Operating Region
- Action messages:
  - IN: Route Plan Request
  - OUT: Route Plan Response



Assignment
RoutePlanner
Task
PlanBuilder
AMASE
Interface
Initialization
[Configurations]
[States]
[Airspace]
[Task]
[TaskInit]
Assignment
[AutomationReq]
[TaskOptions]
[CostMatrix]
[AssignmentOrder]
Implementation
[TaskImplReq]
[RouteRequest]
[RouteResponse]
[TaskImplRsp]
[AutomationRsp]
Execution
[State]
[Actions]
[TaskComplete]

Assignment Service

- **Service that does the primary computation to determine an efficient ordering and assignment of all *Tasks* to the available vehicles**
- **Reasons only at the cost level**
- **First finds a feasible solution by executing a depth-first, greedy search then incrementally improves**
- **Necessary background data messages:**
  - **Automation Request**
  - **Task Plan Options**
- **Action messages:**
  - **IN: Cost Matrix**
  - **OUT: Assignment Order**


Interface
Assignment
RoutePlanner
Task
PlanBuilder
AMASE
Initialization
[Configurations]
[States]
[Airspace]
[Task]
[TaskInit]
Assignment
[AutomationReq]
[TaskOptions]
[CostMatrix]
[AssignmentOrder]
Implementation
[TaskImplReq]
[RouteRequest]
[RouteResponse]
[TaskImplRsp]
[AutomationRsp]
Execution
[State]
[Actions]
[TaskComplete]

Plan Builder

- Service that converts the decisions made by the Assignment Service into waypoint paths that can be sent to each of the vehicles
- Queries each *Task* in turn to construct en-route and on-task waypoints to complete the mission
- First finds a feasible solution by executing a depth-first, greedy search then incrementally improves
- Necessary background data messages:
  - Automation Request
  - Task Plan Options
- Action messages:
  - IN: Assignment Order
  - OUT: Automation Response

Complete Assignment Process

- Simplified version only for explanation
- Full sequence diagram at: OpenUxAS/doc/reference/SequenceDiagrams/
- Main differences:
  - Inclusion of a validation step
  - Task manager to create new task services as they enter the system
  - Orchestration service for handling multiple route planners and sequencing route planning requests
  - Waypoint manager for serving limited sets of wps
  - AMASE is one possible vehicle interface, can be replaced with adapter to translate to other protocol
- Detailed description at OpenUxAS Wiki

Questions?