Core Services Description

Derek Kingston edited this page an hour ago · 16 revisions

This is the pretty-printed version of: ./doc/reference/UserManual/Services/CoreServices.md

This wikipage was last updated: 2017-05-27, 11:38am.

Table of Contents

List of Services

Primary

- AssignmentTreeBranchBoundService
- AutomationRequestValidatorService
- PlanBuilderService
- RouteAggregatorService
- RoutePlannerVisibilityService
- TaskManager

Extended / Derived

- OperatingRegionStateService
- OsmPlannerService
- SensorManagerService
- ServiceManager
- WaypointPlanManagerService

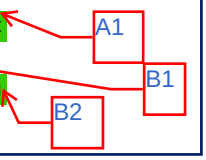
Service and Message Descriptions

AutomationRequestValidatorService

This service provides a simple sanity check on external automation requests. It also queues requests and tags them with unique identifiers, feeding them into the system one at a time.

This service has two states: **idle** and **busy**. In both states, when a non *AutomationRequest* message is received, a local memory store is updated to maintain a list of all available tasks, vehicle configurations, vehicle states, zones, and operating regions.

Upon reception of an *AutomationRequest* message, this service ensures that such a request can be carried out by checking its local memory store for existence of the requested vehicles, tasks, and operating region. If the request includes vehicles, tasks, or an operating region that has not previously been defined, this service will publish an error message.



Upon determination that the *AutomationRequest* includes only vehicles, tasks, and an operating region that have previously been defined, this service creates a *UniqueAutomationRequest* with a previously unused unique identifier. If in the **idle** state, this service will immediately publish the *UniqueAutomationRequest* message and transition to the **busy** state. If already in the **busy** state, the *UniqueAutomationRequest* will be added to the end of a queue.

When this service receives either an error message (indicating that the *UniqueAutomationRequest* cannot be fulfilled or a corresponding *UniqueAutomationResponse*), it will publish the same message. If in the **idle** state, it will remain in the **idle** state. If in the **busy** state, it will remove from the queue the request that was just fulfilled and then send the next *UniqueAutomationRequest* in the queue. If the queue is empty, this service transitions back to the **idle** state.

This service also includes a parameter that allows an optional *timeout* value to be set. When a *UniqueAutomationRequest* is published, a timer begins. If the *timeout* has been reached before a *UniqueAutomationResponse* is received, an error is assumed to have occured and this service removes the pending *UniqueAutomationRequest* from the queue and attempts to send the next in the queue or transition to **idle** if the queue is empty.

Received messages

Table of messages that the *AutomationRequestValidatorService* receives and processes.

Message Subscription	Description

AutomationRequest (1 ms work)	Primary message to request a set of Tasks to be completed by a set of vehicles in a particular airspace configuration (described by an *OperatingRegion*).
EntityConfiguration (0 ms work)	Vehicle capabilities (e.g. allowable speeds) are described by entity configuration messages. Any vehicle requested in an *AutomationRequest* must previously be described by an associated *EntityConfiguration*.
EntityState (0 ms work)	Describes the actual state of a vehicle in the system including position, speed, and fuel status. Each vehicle in an *AutomationRequest* must have reported its state.
Task (1 ms work)	Details a particular task that will be referenced (by ID) in an *AutomationRequest*.
TaskInitialized (0 ms work)	Indicates that a particular task is ready to proceed with the task assignment sequence. Each task requested in the *AutomationRequest* must be initialized before a *UniqueAutomationRequest* is published.
KeepOutZone (0 ms work)	Polygon description of a region in which vehicles must not travel. If referenced by the *OperatingRegion* in the *AutomationRequest*, zone must exist for request to be valid.
KeepInZone (0 ms work)	Polygon description of a region in which vehicles must remain during travel. If referenced by the *OperatingRegion* in the *AutomationRequest*, zone must exist for request to be valid.
OperatingRegion (1 ms work)	Collection of *KeepIn* and *KeepOut* zones that describe the allowable space for vehicular travel. Must be defined for *AutomationRequest* to be valid.
UniqueAutomationResponse (1 ms work)	Completed response from the rest of the task assignment process. Indicates that the next *AutomationRequest* is ready to be processed.

Published messages

Table of messages that the *AutomationRequestValidatorService* publishes.

Message Publication	Description
UniqueAutomationRequest	A duplicate message to an external *AutomationRequest* but only published if the request is determined to be valid. Also includes a unique identifier to match to the corresponding response.
ServiceStatus	Error message when a request is determined to be invalid. Includes human readable error message that highlights which portion of the *AutomationRequest* was invalid.
AutomationResponse	Upon reception of a completed *UniqueAutomationResponse*, this message is published as a response to the original request.

TaskManagerService

The *TaskManagerService* is a very straight-forward service. Upon reception of a Task message, it will send the appropriate *CreateNewService* message. To do so, it catalogues all entity configurations and current states; areas, lines, and points of interest; and current waypoint paths for each vehicle. This information is stored in local memory and appended as

part of the *CreateNewService* message which allows new Tasks to immediately be informed of all relevant information needed to carry out a Task.

When *TaskManagerService* receives a *RemoveTasks* message, it will form the appropriate *KillService* message to properly destroy the service that was created to fulfill the original Task.

Received messages

Table of messages that the *TaskManagerService* receives and processes.

Message Subscription	**Description**
Task (1 ms work)	Primary message that describes a particular task. The task manager will make the appropriate service creation message to build a service that directly handles this requested Task.
RemoveTasks (1 ms work)	Indicates that Task is no longer needed and will not be included in future *AutomationRequest* messages. Task manager will send the proper *KillService* message to remove the service that was constructed to handle the requested Task.
EntityConfiguration (0 ms work)	Vehicle capabilities (e.g. allowable speeds) are described by entity configuration messages. New Tasks are informed of all known entities upon creation.
EntityState (0 ms work)	Describes the actual state of a vehicle in the system including position, speed, and fuel status. New Tasks are informed of all known entity states upon creation.
AreaOfInterest *LineOfInterest* *PointOfInterest* (0 ms work)	Describes known geometries of areas, lines, and points. New Tasks are informed of all such named areas upon creation.
MissionCommand (0 ms work)	Describes current set of waypoints that a vehicle is following. New Tasks are informed of all known current waypoint routes upon creation.

Published messages

Table of messages that the *TaskManagerService* publishes.

Message Publication	**Description**
CreateNewService	Primary message published by the Task Manager to dynamically build a new Task from an outside description of such a Task.
KillService	When Tasks are no longer needed, the Task Manager will correctly clean up and destroy the service that was built to handle the original Task.

Task

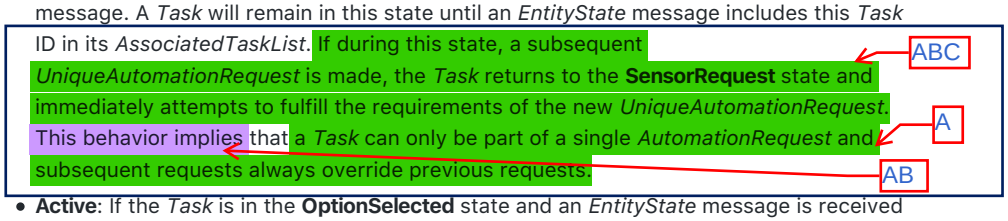
A *Task* forms the core functionality of vehicle behavior. It is the point at which a vehicle (or set of vehicles) is dedicated to a singular goal. During *Task* execution, a wide spectrum of behavior is allowed, including updating waypoints and steering sensors. As part of the core services, this general *Task* description stands in for all *Tasks* running in the system.

The general *Task* interaction with the rest of the task assignment pipeline is complex. It is the aggregation of each *Task's* possibilities that defines the complexity of the overall mission assignment. These *Task* possibilities are called *options* and they describe the

precise ways that a *Task* could unfold. For example, a *LineSearchTask* could present two options to the system: 1) search the line from East-to-West and 2) search the line from West-to-East. Either is valid and a selection of one of these options that optimizes overall mission efficiency is the role of the assignment service.

A general *Task* is comprised of up to nine states with each state corresponding to a place in the message sequence that carries out the task assignment pipeline. The states for a *Task* are:

- **Init**: This is the state that all *Tasks* start in and remain until all internal initialization is complete. For example, a *Task* may need to load complex terrain or weather data upon creation and will require some (possibly significant) start-up time. When a *Task* has completed its internal initialization, it must report transition from this state via the *TaskInitialized* message.
- **Idle**: This represents the state of a *Task* after initialization, but before any requests have been made that include the *Task*. *UniqueAutomationRequest* messages trigger a transition from this state into the **SensorRequest** state.
- **SensorRequest**: When a *Task* is notified of its inclusion (by noting the presence of its ID in the *Tasks* list of an *UniqueAutomationRequest* message), it can request calculations that pertain to the sensors onboard the vehicles that are also included in the *UniqueAutomationRequest* message. While waiting for a response from the *SensorManagerService*, a *Task* is in the **SensorRequest** state and will remain so until the response from the *SensorManagerService* is received.
- **OptionRoutes**: After the *SensorManagerService* has replied with the appropriate sensor calculations, the *Task* can request waypoints from the *RouteAggregatorService* that carry out the on-*Task* goals. For example, an *AreaSearchTask* can request routes from key surveillance positions that ensure sensor coverage of the entire area. The *Task* remains in the **OptionRoutes** state until the *RouteAggregatorService* replies.
- **OptionsPublished**: When routes are returned to the *Task*, it will utilize all route and sensor information to identify and publish the applicable *TaskOptions*. The determination of *TaskOptions* is key to overall mission performance and vehicle behavior. It is from this list of options that the assignment will select in order to perform this particular *Task*. After publication of the options, a *Task* waits in the **OptionsPublished** state until the *TaskImplementationRequest* message is received, whereupon it switches to **FinalRoutes**.
- **FinalRoutes**: Upon reception of a *TaskImplementationRequest*, a *Task* is informed of the option that was selected by the assignment service. At this point, a *Task* must create the final set of waypoints that include both *enroute* and *on-task* waypoints from the specified vehicle location. The *Task* is required to create the *enroute* waypoints since a route refinement is possible, taking advantage of the concrete prior position of the selected vehicle. The *Task* remains in the **FinalRoutes** state until the route request is fulfilled by the *RouteAggregatorService*.
- **OptionSelected**: When the final waypoints are returned from the *RouteAggregatorService*, the *Task* publishes a complete *TaskImplementationResponse* message. A *Task* will remain in this state until an *EntityState* message includes this *Task* ID in its *AssociatedTaskList*. If during this state, a subsequent *UniqueAutomationRequest* is made, the *Task* returns to the **SensorRequest** state and immediately attempts to fulfill the requirements of the new *UniqueAutomationRequest*. This behavior implies that a *Task* can only be part of a single *AutomationRequest* and subsequent requests always override previous requests.



- **Active**: If the *Task* is in the **OptionSelected** state and an *EntityState* message is received which includes the *Task* ID in the *AssociatedTaskList*, then the *Task* switches to the **Active** state and is allowed to publish new waypoints and sensor commands at will. A *Task* remains in the **Active** state until a subsequent *EntityState* message does *not* list the *Task* ID in its *AssociatedTaskList*. At which point, a transition to **Completed** is made. Note that a *Task* can reliquish control indirectly by sending the vehicle to a waypoint not tagged with its own ID. Likewise, it can maintain control indefinitely by ensuring that the vehicle only ever go to a waypoint that includes its ID. If a *UniqueAutomationRequest*

message that includes this *Task* ID is received in the **Active** state, it transitions to the **Completed** state.

- **Completed**: In this state, the *Task* publishes a *TaskComplete* message and then immediately transitions to the **Idle** state.

Received messages

Table of messages that a general *Task* receives and processes.

Message Subscription	Description
UniqueAutomationRequest (2 ms work)	Indicates which *Tasks* are to be considered as well as the set of vehicles that can be used to fulfill those *Tasks*. Upon reception of this message, if a *Task* ID is included, it will publish *TaskPlanOptions*.
TaskImplementationRequest (2 ms work)	After an assignment has been made, each *Task* involved is requested to build the final set of waypoints that complete the *Task* and corresponding selected option. A *Task* must build the route **to** the *Task* as well as waypoints that implement the *Task*. For each on-task waypoint, the *AssociatedTaskList* must include the *Task* ID.
EntityConfiguration (0 ms work)	Vehicle capabilities (e.g. allowable speeds) are described by entity configuration messages. *Tasks* can reason over sensor and vehicle capabilites to present the proper options to other parts of the system. If a vehicle does not have the capability to fulfill the *Task* (e.g. does not have a proper sensor), then the *Task* shall not include that vehicle ID in the list of eligible entities reported as part of an option.
EntityState (0 ms work)	Describes the actual state of a vehicle in the system including position, speed, and fuel status. This message is primary feedback mechanism used for *Tasks* to switch to an **Active** state. When a *Task* ID is listed in the *AssociatedTaskList* of an *EntityState* message, the *Task* is allowed to update waypoints and sensor commands at will.
RouteResponse (1 ms work)	Collection of route plans that fulfill a set of requests for navigation through an *OperatingRegion*. A *Task* must request the waypoints to route a vehicle from its last to the start of the *Task*. Additionally, this message can be used to obtain on-task waypoints.

Published messages

Table of messages that a general *Task* publishes.

Message Publication	Description
TaskPlanOptions	Primary message published by a *Task* to indicate the potential different ways a *Task* could be completed. Each possible way to fulfill a *Task* is listed as an *option*. *TaskOptions* can also be related to each other via Process Algebra.
TaskImplementationResponse	Primary message published by a *Task* that reports the final set of waypoints to both navigate the selected vehicle to the *Task* as well as the waypoints necessary to complete the *Task* using the selected option.

RouteRequest	Collection of route plan requests to leverage the route planner capability of constructing waypoints that adhere to the designated *OperatingRegion*. This request is made for waypoints en-route to the *Task* as well as on-task waypoints.
VehicleActionCommand	When a *Task* is **Active**, it is allowed to update sensor navigation commands to on-task vehicles. This message is used to directly command the vehicle to use the updated behaviors calculated by the *Task*.
TaskComplete	Once a *Task* has met its goal or if a vehicle reports that it is no longer on-task, a previously **Active** *Task* must send a *TaskComplete* message to inform the system of this change.

RoutePlannerVisibilityService

The *RoutePlannerVisibilityService* is a service that provides route planning using a visibility heuristic. One of the fundamental architectural decisions in UxAS is separation of route planning from task assignment. This service is an example of a route planning service for aircraft. Ground vehicle route planning (based on Open Street Maps data) can be found in the *OsmPlannerService*.

The design of the *RoutePlannerVisibilityService* message interface is 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. The logic necessary to plan for multiple (possibly heterogeneous) vehicles is handled in the *RouteAggregatorService*.

A

AB1-3

In two dimensional environments composed of polygons, the shortest distance between points lies on the visibility graph. The *RoutePlannerVisibilityService* creates such a graph and, upon request, adds desired start/end locations to quickly approximate a distance-optimal route through the environment. With the straight-line route created by the searching the visibility graph, a smoothing operation is applied to ensure that minimum turn rate constraints of vehicles are satisfied. Note, this smoothing operation can violate the prescribed keep-out zones and is not guaranteed to smooth arbitrary straight-line routes (in particular, path segments shorter than the minimum turn radius can be problematic).

Due to the need to search over many possible orderings of *Tasks* during an assignment calculation, the route planner must very quickly compute routes. Even for small problems, hundreds of routes must be calculated before the assignment algorithm can start searching over the possible ordering. For this reason it is imperative that the route planner be responsive and efficient.

ABCD

ABCDE

ABCDEF

ABC

A

AB

Received messages

Table of messages that the *RoutePlannerVisibilityService* receives and processes.

Message Subscription	**Description**
RoutePlanRequest (10 ms work)	Primary message that describes a route plan request. A request considers only a single vehicle in a single *OperatingRegion* although it can request multiple pairs of start and end locations with a single message.
KeepOutZone (0 ms work)	Polygon description of a region in which vehicles must not travel. This service will track all *KeepOutZones* to compose them upon reception of an *OperatingRegion*.
KeepInZone (0 ms work)	Polygon description of a region in which vehicles must remain during travel. This service will track all *KeepInZones* to

	compose them upon reception of an *OperatingRegion*.
OperatingRegion (20 ms work)	Collection of *KeepIn* and *KeepOut* zones that describe the allowable space for vehicular travel. When received, this service creates a visibility graph considering the zones referenced by this *OperatingRegion*. Upon *RoutePlanRequest* the visibility graph corresponding to the *OperatingRegion* ID is retreived and manipulated to add start/end locations and perform the shortest path search.
EntityConfiguration (20 ms work)	Vehicle capabilities (e.g. allowable speeds) are described by entity configuration messages. This service calculates the minimum turn radius of the entity by using the max bank angle and nominal speed. Requested routes are then returned at the nominal speed and with turns approximating the minimum turn radius.
"AircraftPathPlanner" (10 ms work)	In addition to subscribing to the above broadcasted messages, this service also subscribes to the group mailbox for path planners that service aircraft requests. Upon reception of a message on this channel, the service will check for one of the above messages and process it as if it came from over the broadcast channel. In either case, the return message always uses return-to-sender addressing.

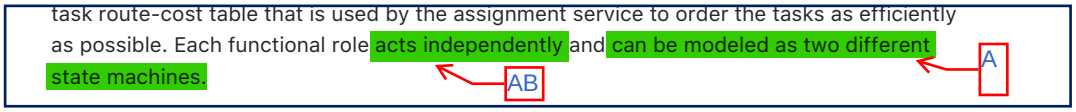
Published messages

Table of messages that the *RoutePlannerVisibilityService* publishes.

Message Publication	**Description**
RoutePlanResponse	This message contains the waypoints and time cost that fulfills the route request. This message is the only one published by the *RoutePlannerVisibilityService* and is always sent using the return-to-sender addressing which ensures that only the original requester receives the response.

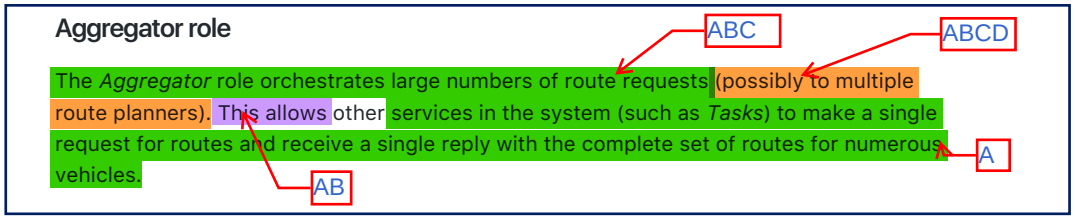
RouteAggregatorService

The *RouteAggregatorService* fills two primary roles: 1) it acts as a helper service to make route requests for large numbers of heterogenous vehicles; and 2) it constructs the task-to-task route-cost table that is used by the assignment service to order the tasks as efficiently as possible. Each functional role acts independently and can be modeled as two different state machines.



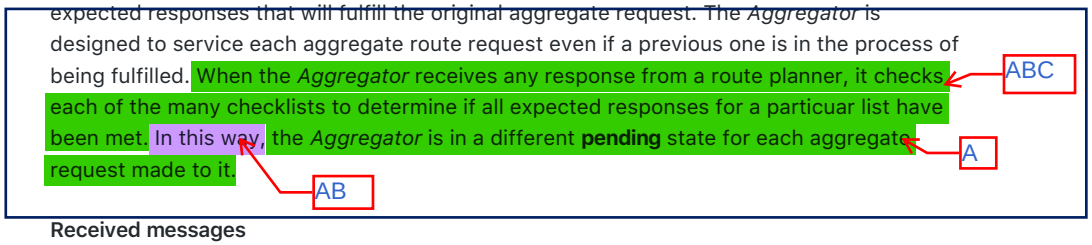
Aggregator role

The *Aggregator* role orchestrates large numbers of route requests (possibly to multiple route planners). This allows other services in the system (such as *Tasks*) to make a single request for routes and receive a single reply with the complete set of routes for numerous vehicles.



For every aggregate route request (specified by a *RouteRequest* message), the *Aggregator* makes a series of *RoutePlanRequests* to the appropriate route planners (i.e. sending route plan requests for ground vehicles to the ground vehicle planner and route plan requests for aircraft to the aircraft planner). Each request is marked with a request ID and a list of all request IDs that must have matching replies is created. The *Aggregator* then enters a **pending** state in which all received plan replies are stored and then checked off the list of expected replies. When all of the expected replies have been received, the *Aggregator* publishes the completed *RouteResponse* and returns to the **idle** state.

Note that every aggregate route request corresponds to a separate internal checklist of expected responses that will fulfill the original aggregate request. The *Aggregator* is designed to service each aggregate route request even if a previous one is in the process of being fulfilled. When the *Aggregator* receives any response from a route planner, it checks each of the many checklists to determine if all expected responses for a particuar list have been met. In this way, the *Aggregator* is in a different **pending** state for each aggregate request made to it.



Received messages

Table of messages that the *RouteAggregatorService* receives and processes in its *Aggregator* role.

Message Subscription	Description
RouteRequest (3 ms work)	Primary message that requests a large number of routes for potentially heterogeneous vehicles. The *Aggregator* will make a series of *RoutePlanRequests* to the appropriate planners to fulfill this request.
EntityConfiguration (0 ms work)	Vehicle capabilities (e.g. allowable speeds) are described by entity configuration messages. This service uses the *EntityConfiguration* to determine which type of vehicle corresponds to a specific ID so that ground planners are used for ground vehicles and air planners are used for aircraft.
RoutePlanResponse (1 ms work)	This message is the fulfillment of a single vehicle route plan request which the *Aggregator* catalogues until the complete set of expected responses is received.

Published messages

Table of messages that the *RouteAggregatorService* publishes in its *Aggregator* role.

Message Publication	Description
RouteResponse	Once the *Aggregator* has a complete set of responses collected from the route planners, the message is built as a reply to the original *RouteRequest*.
RoutePlanRequest	The *Aggregator* publishes a series of these requests in order to fulfill an aggregate route request. These messages are published in batch, without waiting for a reply. It is expected that eventually all requests made will be fulfilled.

Collector role

The *RouteAggregatorService* also acts in the role of creating the *AssignmentCostMatrix* which is a key input to the assignment service. For simplicity, this role will be labeled as the *Collector* role. This role is triggered by the *UniqueAutomationRequest* message and begins the process of collecting a complete set of on-task and between-task costs.

The *Collector* starts in the **Idle** state and upon reception of a *UniqueAutomationRequest* message, it creates a list of *Task* IDs that are involved in the request and then moves to the **OptionsWait** state. In this state, the *Collector* stores all *TaskPlanOptions* and matches them to the IDs of the *Task* IDs that were requested in the *UniqueAutomationRequest*. When the expected list of *Tasks* is associated with a corresponding *TaskPlanOptions*, the *Collector* moves to the **RoutePending** state. In this state, the *Collector* makes a series of route plan requests from 1) initial conditions of all vehicles to all tasks and 2) route plans between the end of each *Task* and start of all other *Tasks*. Similar to the *Aggregator*, the *Collector* creates a checklist of expected route plan responses and uses that checklist to determine

when the complete set of routes has been returned from the route planners. The *Collector* remains in the **RoutePending** state until all route requests have been fulfilled, at which point it collates the responses into a complete *AssignmentCostMatrix*. The *AssignmentCostMatrix* message is published and the *Collector* returns to the **Idle** state.

Note that the *AutomationValidatorService* ensures that only a single *UniqueAutomationRequest* is handled by the system at a time. However, the design of the *Collector* does allow for multiple simultaneous requests as all checklists (for pending route and task option messages) are associated with the unique ID from each *UniqueAutomationRequest*.

Received messages

Table of messages that the *RouteAggregatorService* receives and processes in its *Collector* role.

Message Subscription	Description
UniqueAutomationRequest (1 ms work)	Primary message that initiates the collection of options sent from each *Task* via the *TaskPlanOptions* message. A list of all *Tasks* included in the *UniqueAutomationRequest* is made upon reception of this message and later used to ensure that all included *Tasks* have responded.
TaskPlanOptions (2 ms work)	Primary message from *Tasks* that prescribe available start and end locations for each option as well as cost to complete the option. In the **RoutePending** state, the *Collector* will use the current location of the vehicle to create paths from each vehicle to each task option and from each task option to every other task option.
EntityState (0 ms work)	Describes the actual state of a vehicle in the system including position, speed, and fuel status. This message is used to create routes and cost estimates from the associated vehicle position and heading to the task option start locations.
RoutePlanResponse (2 ms work)	This message is the fulfillment of a single vehicle route plan request which the *Collector* catalogues until the complete set of expected responses is received.

Published messages

Table of messages that the *RouteAggregatorService* publishes in its *Collector* role.

Message Publication	Description
AssignmentCostMatrix	Once the *Collector* has a complete set of *TaskPlanOptions* as well as routes between tasks and vehicles, this message is built to inform the next step in the task assignment pipeline: the *AssignmentTreeBranchBoundService*.
RoutePlanRequest	The *Collector* publishes a series of these requests in order to compute the vehicle-to-task and task-to-task route costs. These messages are published in batch, without waiting for a reply. It is expected that eventually all requests made will be fulfilled.

AssignmentTreeBranchBoundService

The *AssignmentTreeBranchBoundService* is a service that does the primary computation to determine an efficient ordering and assignment of all *Tasks* to the available vehicles. The assignment algorithm reasons only at the cost level; in other words, the assignment itself

does not directly consider vehicle motion but rather it uses estimates of that motion cost. The cost estimates are provided by the *Tasks* (for on-task costs) and by the *RouteAggregatorService* for task-to-task travel costs.

A = It is unlikely to request a mission for which no feasible solution exists

The *AssignmentTreeBranchBoundService* can be configured to optimize based on cumulative team cost (i.e. sum total of time required from each vehicle) or the maximium time of final task completion (i.e. only the final time of total mission completion is minimized). For either optimization type, this service will first find a feasible solution by executing a depth-first, greedy search. Although it is possible to request a mission for which **no** feasible solution exists, the vast majority of missions are underconstrained and have an exponential (relative to numbers of vehicles and tasks) number of solutions from which an efficient one must be discovered.

AAAA

AAAAA

After the *AssignmentTreeBranchBoundService* obtains a greedy solution to the assignment problem, it will continue to search the space of possibilities via backtracking up the tree of possibilities and *branching* at descision points. The cost of the greedy solution acts as a *bound* beyond which no solution is be considered. In other words, as more efficient solutions are discovered, any partial solution that exeeds the cost of the current best solution will immediately be abandoned (cut) to focus search effort in the part of the space that could possibly lead to better solutions. In this way, solution search progresses until all possibilities have been exhausted or a pre-determined tree size has been searched. By placing an upper limit on the size of the tree to search, worst-case bounds on computation time can be made to ensure desired responsiveness from the *AssignmentTreeBranchBoundService*.

AAABA

AAAB

AAABA

AAABB

AA

AAA

A

General assignment problems do not normally allow for specification of *Task* relationships. However, the *AssignmentTreeBranchBoundService* relies on the ability to specify *Task* relationships via Process Algebra constraints. This enables creation of moderately complex missions from simple atomic *Tasks*. Adherence to Process Algebra constraints also allows *Tasks* to describe their *option* relationships. The Process Algebra relationships of a particular *Task* option are directly substituted into and replace the original *Task* in the mission-level Process Algebra specification. Due to the heavy reliance on Process Algebra specifications, any assignment service that replaces *AssignmentTreeBranchBoundService* must also guarantee satisfaction of such specifications.

The behavior of the *AssignmentTreeBranchBoundService* is straight-foward. Upon reception of a *UniqueAutomationRequest*, this service enters the **wait** state and remains in this state until a complete set of *TaskPlanOptions* and an *AssignmentCostMatrix* message have been received. In the **wait** state, a running list of the expected *TaskPlanOptions* is maintained and checked off when received. Upon receiving the *AssignmentCostMatrix* (which should be received strictly after the *TaskPlanOptions* due to the behavior of the *RouteAggregatorService*), this service conducts the branch-and-bound search to determine the proper ordering and assignment of *Tasks* to vehicles. The results of the optimization are packaged into the *TaskAssignmentSummary* and published, at which point this service returns to the **idle** state.

Received messages

Table of messages that the *AssignmentTreeBranchBoundService* receives and processes.

Message Subscription	Description
UniqueAutomationRequest (0 ms work)	Sentinel message that initiates the collection of options sent from each *Task* via the *TaskPlanOptions* message. A list of all *Tasks* included in the *UniqueAutomationRequest* is made upon reception of this message and later used to ensure that all included *Tasks* have responded.
TaskPlanOptions (0 ms work)	Primary message from *Tasks* that prescribe available start and end locations for each option as well as cost to complete the option. In the **wait** state, this service will store all reported options for use in calculating mission

	cost for vehicles when considering possible assignments.
AssignmentCostMatrix (1500 ms work)	Primary message that initiates the task assignment optimization. This message contains the task-to-task routing cost estimates and is a key factor in determining which vehicle could most efficiently reach a *Task*. Coupled with the on-task costs captured in the *TaskPlanOptions*, a complete reasoning over both traveling to and completing a *Task* can be looked up during the search over possible *Task* orderings.

Published messages

Table of messages that the *AssignmentTreeBranchBoundService* publishes.

Message Publication	Description
TaskAssignmentSummary	The singular message published by this service which precisely describes the proper ordering of *Tasks* and the vehicles that are assigned to complete each *Task*.

PlanBuilderService

The final step in the task assignment pipeline is converting the decisions made by the 'AssignmentTreeBranchBoundService *into waypoint paths that can be sent to each of the vehicles. Using the ordering of* Tasks *and the assigned vehicle(s) for each* Task*, the* PlanBuilderService *will query each* Task *in turn to construct enroute and on-task waypoints to complete the mission.*

Similar to both the *RouteAggregator* and the *AssignmentTreeBranchBoundService*, the *PlanBuilderService* utilizes a received *UniqueAutomationRequest* to detect that a new mission request has been made to the system. The *UniqueAutomationRequest* is stored until a *TaskAssignmentSummary* that corresponds to the unique ID is received. At this point, the *PlanBuilderService* transitions from the *idle* state to the *busy* state.

Using the list of ordered *Tasks* dictated by the *TaskAssignmentSummary*, the *PlanBuilderService* sends a *TaskImplementationRequest* to each *Task* in order and waits for a *TaskImplementationResponse* from each *Task* before moving to the next. This is necessary as the ending location of a previous *Task* becomes the starting location for a subsequent *Task*. Since each *Task* is allowed to refine its final waypoint plan at this stage, the exact ending location may be different that was was originally indicated during the *TaskPlanOptions* phase. By working through the *Task* list in assignment order, all uncertainty about timing and location is eliminated and each *Task* is allowed to make a final determination on the waypoints to be used.

Once all *Tasks* have reponded with a *TaskImplementationResponse*, the *PlanBuilderService* links all waypoints for each vehicle into a complete *MissionCommand*. The total set of *MissionCommands* are collected into the *UniqueAutomationResponse* which is broadcast to the system and represents a complete solution to the original *AutomationRequest*. At this point, the *PlanBuilderService* returns to the *idle* state.

Received messages

Table of messages that the *PlanBuilderService* receives and processes.

Message Subscription	Description
TaskAssignmentSummary (2 ms work)	Primary message that dictates the proper order and vehicle assignment to efficiently carry out the requested mission. Upon reception of this messsage, the *PlanBuilderService* queries each *Task* in order for the final waypoint paths.

EntityState (0 ms work)	
TaskImplementationResponse (2 ms work)	Primary message that each *Task* reports to inform this service of the precise waypoints that need to be followed to reach the *Task* and carry it out correctly. The ordered collection of these messages are used to build the final *UniqueAutomationResponse*.
UniqueAutomationRequest (0 ms work)	Informs this service of a new mission request in the system. Contains the desired starting locations and headings of the vehicles that are to be considered as part of the solution.

Published messages

Table of messages that the *PlanBuilderService* publishes.

Message Publication	Description
TaskImplementationRequest	The primary message used to query each *Task* for the proper waypoints that both reach and carry out the *Task*. Once the *PlanBuilderService* receives a corresponding response from each *Task*, it can construct a final set of waypoints for each vehicle.
UniqueAutomationResponse	This message contains a list of waypoints for each vehicle that was considered during the automation request. This collection of complete waypoints for the team fulfills the original request.

(Template for service descriptions)

XXService

((--add description here--))

Received messages

Table of messages that the *XXService* receives and processes.

Message Subscription	Description
??	??
??	??
??	??

Published messages

Table of messages that the *XXService* publishes.

Message Publication	Description
??	??
??	??

??	??