SCHARP Scenario Planning Algorithms

Marcel Becker
becker@kestrel.edu

Stephen Fitzpatrick
fitzpatrick@kestrel.edu

Douglas R. Smith
smith@kestrel.edu

Abstract

SCHARP is a planning and execution monitoring system for air campaigns developed under the DARPA Resilient Synchronized Planning and Assessment for the Contested Environment (RSPACE) program. This report details its architecture and algorithms.

Kestrel Institute

The results described in this document were supported by DARPA Resilient Synchronized Planning and Assessment for the Contested Environment (RSPACE) Program under Air Force Research Laboratory (AFRL) contract HR0011-15-C-0138.

Contents

Introduction

SCHARP is a planning and execution monitoring system for air operations developed under the DARPA RSPACE program. The planning component takes as input a *scenario* that defines the objectives of the air campaign (e.g., prioritized targets), the resources available (e.g., aircraft and munitions), and constraints (e.g., airspace restrictions). SCHARP produces an *Air Tasking Order* (ATO) that defines the missions the resource will execute to achieve the objectives subject to the constraints. The ATO can be serialized into standard formats — *United States Message Text Format* (USMTF) plain text or *Air Operations Community of Interest* (AOCOI) XML — or a custom JSON format. The data can be obtained from files or from web services.

The primary components of a scenario are:

Air Bases: The physical locations where air units and aircraft are located.

Aircraft Models: The set of parameters defining the available resources used by the planner (e.g., fuel capacity and burn rates).

Standard Configuration Loads (SCLs): Each aircraft type is capable of carrying certain combinations of munitions and equipment (e.g., sensors and external fuel taks). A specific combination is defined by an SCL — e.g., 4 GBUs, 4 AIMs and 2 external fuel tanks.

Air Units: An air unit maintains a group of aircraft — of the same type — at an air base. Multiple units may be located at the same base.

Unit Contracts: Each air unit has a unit contract that defines how many sorties it will provide each day using its available aircraft. (Not all aircraft are always available due to maintenance requirements.) The unit contract also defines time windows during which the sorties can occur, and other constraints.

Airspace Points and Orbits: The geographical coordinates of relevant air space regions like orbits, air refueling tracks, battlespace boundaries, etc.

Targets and Target-Weapon Pairings: The prioritized list of targets (with location, description, etc) and the munitions (type and quantity) recommended for use against them.

Non-Strike Mission Requests: Additional requests for orbit missions (e.g., XATK, FAC, DCA), ground alert (e.g., GATK), or other types of missions (e.g., CAS, ISR).

The planning algorithms in SCHARP use the input data to create a set of air missions to engage targets and to protect air bases and friendly assets. This document describes the algorithms used to create the different types of missions supported by SCHARP.

Planner Overview

The SCHARP Planning Engine, or simply the *Planner,* is a high-performance software system capable of computing a full Air Tasking Order (ATO) comprising hundreds of Air Strike Packages and Missions in a matter of seconds.

The two key components of the Planner implementation are a Distributed Control Architecture that orchestrates a decentralized planning process, and a collection of special-purpose planning algorithms that provide the functionality needed to compute the different types of strike, support, and non-strike missions required to satisfy the tactical objectives defined by the AOD and Joint Integrated Prioritized Target List (JIPTL).

One of the key goals of the DARPA RSPACE program was to support a distributed planning process in which several planning cells, co-located or geographically dispersed, collaborate to generate a plan or Air Tasking Order. To accomplish this, the planner was designed to work with network topologies (e.g., number of planning cells) ranging from a fully centralized to a fully distributed model. A fully centralized planning process is one in which a single human planner using a single computer generates a complete ATO. A fully distributed planning process is one in which the authority for making different planning decisions are assigned to different human planners, using multiple computers. We usually refer to the planning system configuration supporting a centralized planning process as a single node configuration. We call the configuration used to support the distributed planning process, a multi-node configuration.

A Planning Node is an executing instance of the SCHARP Planning Engine, configured with a well-defined set of planning authorities. To be more precise, since the Planner is implemented in Java, a Planning Node is just a Java process running on a single Java Virtual Machine instance. Although it is possible to run several Planning Nodes in the same computer, or even several planning nodes in the same Java Virtual Machine, for presentation purposes, let's assume there is just one Planning Node per physical workstation, and one human planner interacting with each Planning Node via the Graphical User Interface.

Each Planning Node hosts multiple Planning Agents. Planning Agents implement the computational logic that creates the missions and packages that get executed, in order to achieve the desired tactical objectives. The Planning Agents orchestrate the execution of special-purpose planning algorithms to create the strike packages and associated support missions. The different Planning Agents communicate with each other, and with the other components of the SCHARP platform, via a message passing mechanism implemented using the Apache ActiveMQ message broker.

The Planning Node configuration defines what Planning Agents will be hosted in what physical nodes. Planning Agents have different planning authorities and capabilities. They interact with each other to create the different types of missions that will be in the Air Tasking Order.

In the next sections we will provide more details about the Distributed Control Architecture, precisely define what a Planning Agent is, discuss the different types of algorithms used for planning air missions, and describe the input data needed by the planner to compute an Air Tasking Order.

Distributed Control Architecture

In the preceding section, we mentioned that the Planner can be configured to run a planning process ranging from fully centralized to fully distributed. We also introduced the concepts of Planning Node, Planning Agent, and communication via message broker. In this section we will explain in more details what these Planning Agents are, how Planning Nodes are configured, and how these agents communicate with each other to generate a plan.

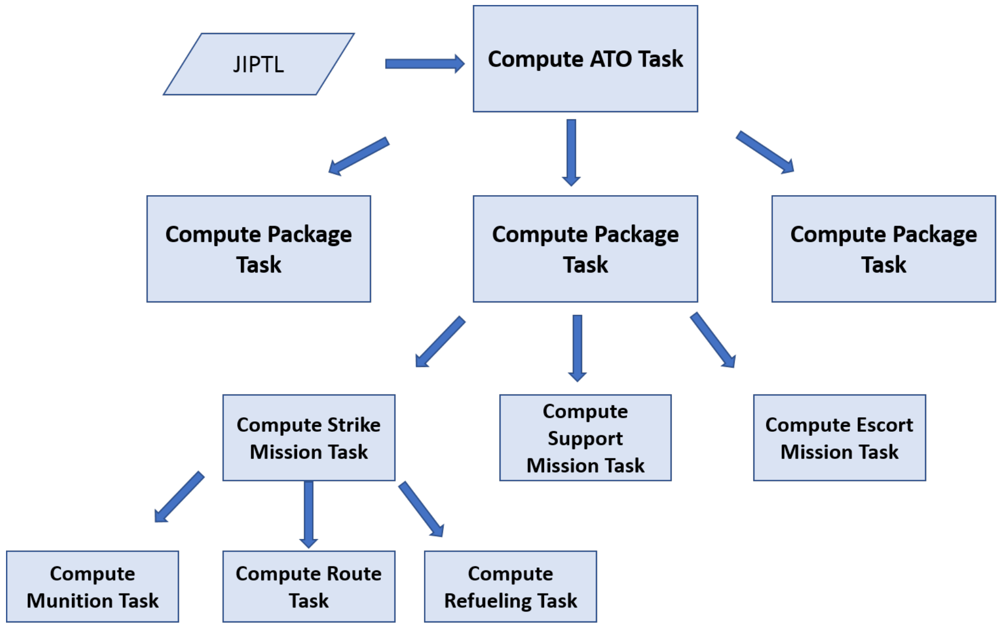

Figure 1: Example Task Decomposition

The key algorithmic concept, or assumption, driving the distributed problem-solving process implemented by the SCHARP Planner is the hierarchical decomposition of tasks, as presented in Figure 1. A *task* is an entity representing a request to generate a full plan, a partial plan, or just a single mission. Larger planning tasks can be decomposed into smaller planning tasks. Plans computed to satisfy smaller tasks can be composed to satisfy larger ones. For example, the generation of an ATO can be decomposed into a number of tasks to generate strike packages to hit the targets in the JIPTL. A strike package request can be further decomposed into tasks for planning one or more strike missions, tasks for planning support missions, tasks for planning air refueling, tasks for planning recon, etc. The plan snippets computed to satisfy a certain task, can then be combined, or composed, to generate larger plans that satisfy the higher-level tasks. The decomposition of tasks into sub-tasks, and the composition of plan snippets into larger plans, and finally into an Air Tasking Order is the fundamental element of the distributed control architecture implemented by the SCHARP Planner.

The Planning Agents are the computational entities (e.g., Java objects) responsible for processing tasks and producing plans. Since tasks are decomposed hierarchically, the Planning Agents are also organized into a planning hierarchy. Agents higher in the hierarchy create tasks and assign them to lower level agents. Agents lower in the hierarchy compute candidate missions for the tasks and send those candidate missions back to the requesting agent. We call the candidate plan generated by an agent in response to a task, a "mission bid" or just a "bid."

Tasks and bids are transmitted among agents using the ActiveMQ (AMQ) message broker. Each agent encodes a task or a bid as an AMQ message and sends it to the corresponding receiver agent. The AMQ broker is just another Java object that can be hosted either (1) in the same Java Virtual Machine running the Planning Node; or (2) in its own, dedicated Java Virtual Machine in the same host as the Planning Node; or (3) in a separated computer reachable from the computer hosting the Planning Node(s).

The AMQ broker takes care of receiving messages from and sending messages to the correct Planning Agent. All the interactions between agents only happens through the AMQ. Given that all interactions between agents are mediated by the AMQ, an agent can be hosted in any physical node, provided the node can reach the AMQ broker.

The Planning Agent structure, and the use of the AMQ broker are the key architectural elements that allow us to configure the planner as a single node or as multiple nodes.

In a single node configuration, all planning agents are hosted in the same Java process but are still communicating via AMQ messages. The fully distributed configuration is a topology in which each planning node hosts a single planning agent. Any topology ranging from a single node hosting all agents, to the fully distributed configuration with one agent per node, is achievable using the implemented SCHARP Distributed Control Architecture.

Planning Agents

A Planning Agent is a computational entity, or a Java object, that takes a task as input and produces a plan candidate as output.

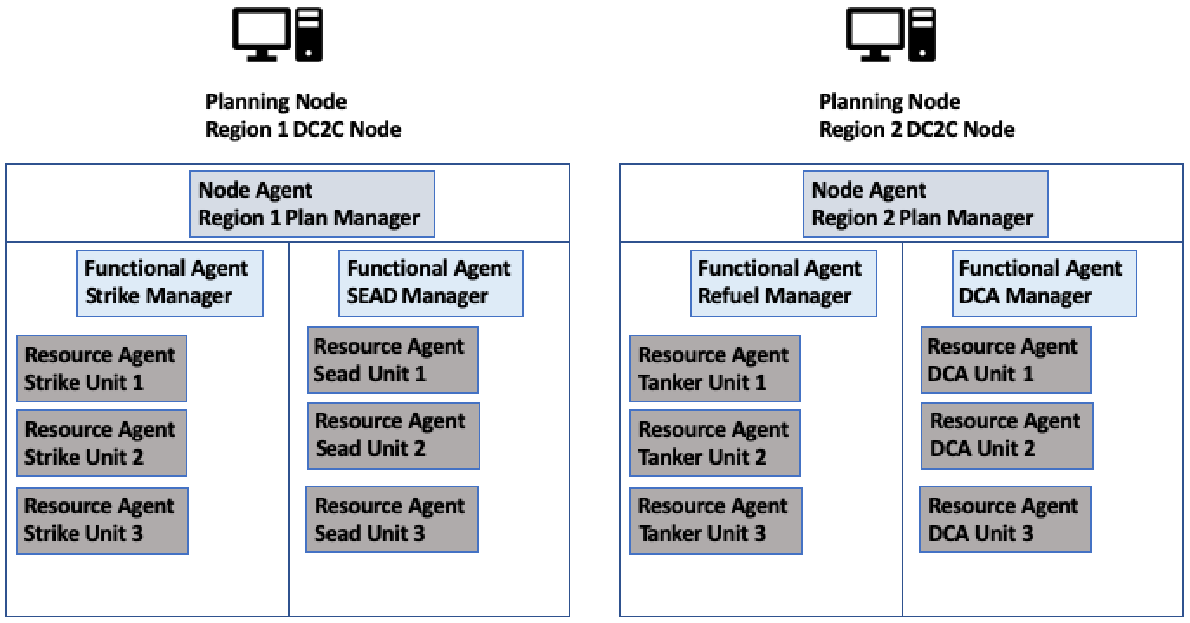

Figure 2: Planner Nodes and Agents

In SCHARP, there are 3 types of planning agents as presented in Figure 2: Node Agent, Functional Agent, and Resource Agent. The Node Agent is the process that bootstraps the Planning Node, and initializes all the other Planning Agents running in a Planning Node. There is only one Node Agent per node.

The Node Agent is responsible for reading the node configuration, instantiating all the other agents hosted in the node, creating the top-level tasks necessary to compute the plan, assigning tasks to other planning agents, collecting the top-level bids generated by other Planning Agents, and composing the plan output.

In a multi-node configuration, the Node Agents are also organized hierarchically. Each node has some well-defined set of planning authorities prescribed in a static document called the Distributed Control Plan (DCP). For each set of fully connected Planning Nodes, one, and only one, of the nodes is selected to be the AOC Node. The AOC node agent has the ultimate responsibility for composing and publishing the ATO. The AOC Node may send tasks to be processed by other Node Agents, but it has the authority to compose and publish the ATO.

All the communication among Planning Nodes goes through the Node Agents. Conceptually, the architecture allows any agent to communicate with any other agent via AMQ. This, however, generates a large amount of network traffic. To reduce the number of messages flowing through an external AMQ broker, we optimized the messaging mechanism, and restricted inter-node messages to always go through the Node Agent: Only Node Agents can send and receive messages from other Planning Nodes. Messages between agents hosted in the same node go through an internal message queue that does not use the external AMQ broker, and therefore is not affected by limited bandwidth. Figure 3 shows an example topology with 4 DC2C Planning Nodes. Each DC2C's Node Agent communicates with other nodes via the message broker. All intra-node communication is handled by an internal message passing mechanisms that avoids using the external AMQ.

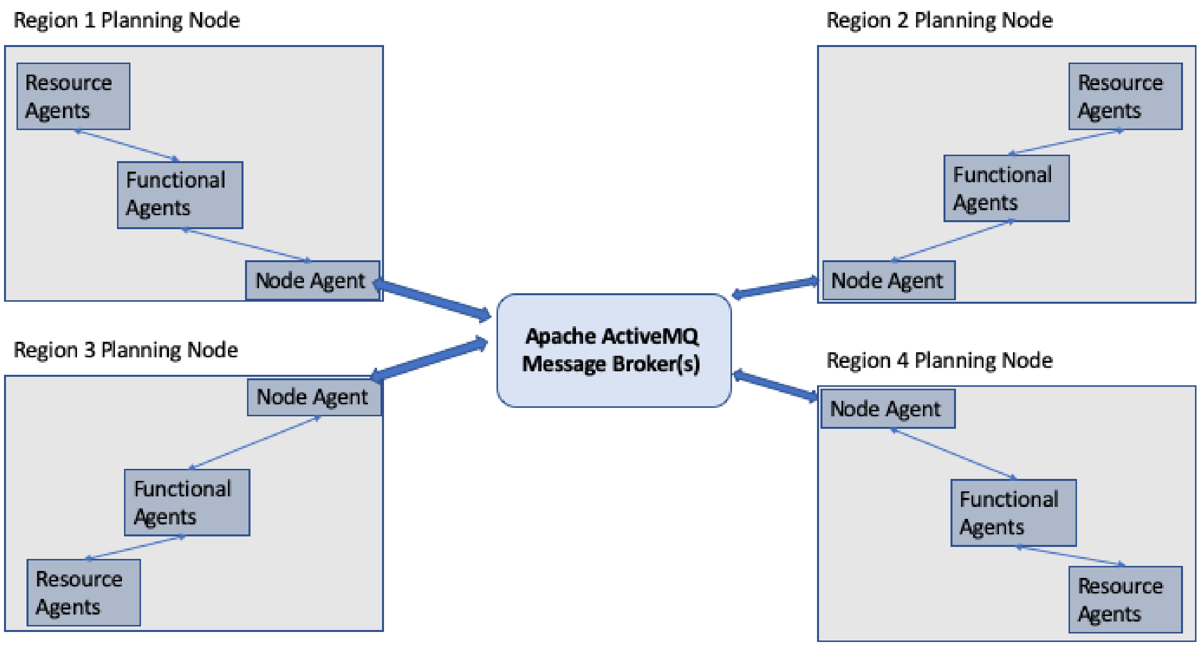

Figure 3: Message Flow

The second type of Planning Agent is the Functional Agent. The Functional Agents are responsible for orchestrating the planning process for specific types of missions or packages. For example, the STRIKE Agent would be responsible for planning strike packages to hit targets in the JIPTL; the SEAD Agent would be responsible for providing SEAD/EW missions to support strike packages; the REFUEL Agent would be responsible for planning tanker missions to provide air refueling to all other missions requiring air refueling. Each cluster of connected nodes has only one active Functional Agent for each mission type the system can plan. Accordingly, there is only one STRIKE agent per cluster of connected nodes, only one REFUEL agent per cluster of connected nodes. The Distributed Control Plan defines which physical node is responsible for planning each specific type of mission. If, for example, STRIKE planning and REFUEL planning authorities are assigned to two different nodes, the STRIKE Agent and the REFUEL Agent are instantiated and executed in different nodes. In this case, air-refueling tasks created as a result of planning strike missions would have to be transmitted between physical nodes.

Functional agents communicate both with other functional agents, and with Resource Agents to plan a mission or a package to satisfy a given task. They also communicate with the Node Agent to return bids for top-level tasks originating from the Node Agent. We will revisit the interaction between the different types of agents after we describe the third type of agent, the Resource Agent.

The third class of Planning Agent is the Resource Agent. A Resource Agent corresponds to a physical Air Unit. This agent manages a homogeneous pool of resources, usually a set of aircraft of the same type, which can be used to implement a specific type of mission. The Resource Agent is responsible for computing all the details of a mission using the available resources managed by the agent. The Resource Agent determines if there are enough resources available for a mission, computes the flight path for the mission, identifies the need for air refueling, identifies the need for additional support (e.g., SEAD and OCA support), and creates the detailed mission representation that is used to generate the ATO representation. For each task, if there are available resources, and if the requirements of the task can be satisfied, a Resource Agent computes one or more mission candidates that satisfy the task requirements. If the current agent cannot satisfy the task, an empty plan, or an empty bid, is returned to the requesting agent.

Resource Agents receive tasks from – and send bids to – Functional Agents only. Resource Agents do not communicate directly with other Resource Agents, nor with Node Agents. The Resource Agents "hide" behind their assigned Functional Agent.

A Resource Agent can plan multiple types of mission if the available resources have multi-role capabilities, and if these resources are shared among those roles. If the resources are multi-role, the Resource Agent may communicate with multiple Functional Agents. If the aircraft type managed by the Resource Agent can only fly one type of mission, the corresponding Resource Agent communicates with only one Functional Agent. Similar to the Functional Agent, for a cluster of connected nodes, there is only one instance of a Resource Agent for a given resource pool. The DCP prescribes in what Planning Node the Resource Agent is hosted. Ideally the Resource Agent would be hosted in the same node as the Functional Agent(s) it can talk to. Each Resource Agent uses a number of different planning algorithms customized for the type of mission it can compute. For example, a strike resource agent uses path planning, shortest path, and threat avoidance algorithms to compute feasible routes for a strike mission; it uses temporal constraint propagation to define start and end times of flight segments, it uses some light weight optimization algorithms to determine the appropriate type and number of munitions to use, etc. A refuel resource agent uses different types of algorithms to create a Tanker mission to make fuel available at a certain air-refueling track and manage the amount of fuel available at each refueling track.

The three types of agents and the message passing mechanism are the basic building blocks of the Distributed Control Architecture. We are now ready to dive into the details of the message and control flow used to plan missions and compose an ATO.

The Distributed Planning Control and Message Flow

The SCHARP Planning Engine runs as a cluster of interconnected Planning Nodes. The Planning Nodes in the cluster communicate via message passing, mediated by an ActiveMQ message broker. The architecture does not dictate how many Planning Nodes should be in a cluster. The cluster size can range from one single node hosting all Planning Agents, to a cluster with as many nodes as Planning Agents available. The cluster should also include one or more instances of the Apache ActiveMQ message broker. All inter-node communication goes through the AMQ instances. If several AMQ instances are present, the instances should be fully connected. The Distributed Control Plan document defines the cluster configuration: It specifies how many nodes are used in a cluster, the unique id for each node, which node hosts the AOC Node Agent, and in what nodes each Functional and Resource Agent are to be hosted. Additional input defines what we call the scenario the planner actually plans. The scenario is a large data set that includes which air bases, air units, and resource types are available, the different resource models, how many resources of what type are available at each unit, the prioritized set of targets that should be struck, the air space configuration, and additional requests for non-strike missions. The Resource Agents correspond to the Air Units described in the input data and in the DCP. A single physical Air Unit may map to multiple Resource Agents if the pool of resources from an Air Unit is split into smaller, dedicated resource pools (e.g., use 8 of the 16 aircraft from an Air Unit for strike missions, and the remaining 8 for XATK missions).

Each Planning Node has a unique identifier provided as an input parameter to the node start up script or command line. When the Planning Node process starts, the bootstrap code loads the scenario data, and instantiates the unique Node Agent for that node. The Node Agent then parses the DCP and initializes all the Functional and Resource Agents hosted in the node. In single node mode, the Node Agent instantiates all the Functional Agents, and all the Resource Agents defined in the input scenario data. After node initialization, the AOC Node is ready to receive user commands, and all the remaining nodes are listening for AMQ messages.

User commands are also transmitted to the AOC Node via AMQ messages. The AOC Node instantiates a special message server that listens for user commands. The user, via the Graphical Web Based Interface, can then start the planning process. The user can ask the system to plan one mission at a time, or to batch process all targets and generate the full plan.

The server listening to user commands forwards the request to the AOC Node Agent. The AOC Agent then processes the request by translating the user requests into planning tasks, packing these tasks into AMQ messages, and sending messages to the Functional or Node agents capable of processing the tasks. After all tasks have been processed, the AOC Node sends a response to the user. The set of planning commands available to the user and the corresponding response messages are defined elsewhere. In this section, we are interested only in the control and message flow after the AOC node receives a user command.

Let's assume the user command is a request to automatically generate a plan to hit all targets in the JIPTL, plan all non-strike missions, and to produce an Air Tasking Order in USMTF format. This command triggers the planning process in fully automated mode. In this case, the Planner generates the entire plan without any guidance from the user. The user can later edit the plan.

After receiving this user request, the AOC Node creates strike tasks for targets in the JIPTL, and tasks for the different types of non-strike missions specified in the AOD (e.g., CAS missions, XATK missions, XEW missions, ISR missions, FAC missions, GATK missions, etc.) A strike task can specify a single target, or a cluster of targets to hit. The AOC Node may pre-process the input data and provide some additional guidance to the other planning agents on how to compute the plan. For example, based on the location of targets and air bases, the AOC agent may pre-define some entry and exit points, the AOC Agent may estimate fuel demand, pre-compute a tanker allocation profile, and create tanker tasks to instruct tanker agents to create these fuel availability profiles. We call the tasks generated by the AOC Agent top-level tasks.

The AOC Agent then prioritizes the tasks and assigns them to the corresponding Functional Agent based on the task type. A strike task is assigned to the STRIKE Functional Agent, a tanker task is assigned to the REFUEL Functional Agent, and so on.

The AOC Node packs each top-level task into an AMQ message and sends the message to the appropriate Functional Manager. If the Functional Manager is not hosted in the same node as the AOC Agent, special messaging code takes care of forwarding the message to the Node Agent responsible for the node where the Functional Agent is located. In this case, the receiving Node Agent forwards the message to the appropriate Functional Agent, and acts as a proxy AOC for that Functional Agent.

The Functional Agent finds the best mission(s) that can execute the task according to some pre-defined criteria. The Functional Agent is responsible for putting together a plan snippet that fully satisfies the task. For example, the STRIKE Agent is responsible for creating a package comprised of: (1) strike missions capable of hitting all targets specified in the task, (2) EW and OCA escort support missions if necessary, and (3) appropriate air refueling for all missions in the package. The STRIKE Agent computes this plan snippet by further decomposing a strike task into smaller tasks and assigning those tasks to other Functional Agents or Resource Agents.

The STRIKE Agent starts by requesting candidate strike missions from the different strike Resource Agents. The strike resource agents are the Agents corresponding to the strike Air Units available in the scenario (e.g., fighter and bomber units). The STRIKE Agent sends a strike task to each of the strike Resource Agents it knows about. Each Resource Agent runs its own planning algorithms, and, if possible, creates a strike mission that satisfies the task.

The STRIKE Functional Agent, collects all the bids from the different strike Resource Agents, and composes a strike package that can hit all the targets specified in the task. The strike package can be composed of one or more strike missions. The STRIKE Functional Agent adds missions to the package until all targets are hit with enough munitions. The STRIKE Functional Agent selects one or more of the bids proposed by the different strike Resource Agents and rejects the remaining bids. Once a bid is selected, the resources used for that bid are committed, and considered unavailable and "in use." If the bid is rejected, the resources used in the bid are returned to the pool of available resources.

The STRIKE Functional Agent may go through several rounds of sending tasks to Resource Agents and selecting bids until it defines a satisfactory strike package capable of hitting all targets in the task. The same Resource Agent can provide more than one mission for a package. After the STRIKE Functional Agent is happy with the strike missions in the package, it analyzes the missions to identify additional requirements for SEAD support, OCA escort, or air refueling. The STRIKE Agent applies some analysis algorithms to the missions in the package, and, if necessary, creates SEAD Tasks, OCA Escort Tasks, and/or

Air Refueling Tasks. These new secondary tasks created in response to additional requirements imposed by the computed plan are called sub-tasks. The STRIKE Functional Agent sends each one of these sub-tasks to the appropriate Functional Agent capable of planning missions of that type: SEAD Tasks go to the SEAD Functional Agent, OCA Escort Tasks go to the OCA Functional Agent; and Air Refueling Tasks go to the REFUEL Functional Agent. If any of these Functional Agents are hosted in a remote node, the message processing mechanism takes care of forwarding the message with the task to the appropriate node.

The SEAD, the OCA, and the REFUEL Functional Agents implement a problem solving strategy similar to the one used by the STRIKE Functional Manager: They send a task to each one of the known Resource Agents capable of processing tasks of that type (e.g., SEAD, OCA or Air Refuel Tasks). Each Resource Agent that has available resources and can feasibly satisfy the task sends back a non-empty bid. The Functional Agent collects all the bids provided by Resource Agents, analyzes the missions in the bids, generates and plans additional sub-tasks if necessary (e.g., SEAD or OCA missions require air refueling), selects the best bid, and returns best bid to STRIKE Functional Agent.

Back in the STRIKE Functional Agent, if all sub-tasks (SEAD, AOC, REFUEL) can be successfully planned, the Functional Agent finalizes the package creation and sends back a bid containing the selected package to the AOC Agent.

After receiving a bid for a task, the AOC proceeds to the next task in a prioritized list of tasks. The AOC Agent sends the top-level tasks one at a time to the appropriate Functional Agent and collects the bids returned for that task before proceeding to the next task. If a task cannot be satisfied, an empty bid is returned.

After all tasks have been processed, the AOC Agent triggers the serialization of the ATO in USMTF format. The ATO serialization is then written to a file stored in the file system of the computer or virtual machine hosting the AOC Node. In addition to the USMF format, the plan is also serialized as a JSON string that is sent back to the User Interface, and as an AOCOI XML string that can be sent to other systems like ATOMS. The same workflow is used for non-strike tasks. The AOC node sends a top-level task to the corresponding Functional Manager. The Functional Manager sends tasks to the mission specific Resource Agents. Each Resource Agent responds with a bid that may eventually contain an empty plan if that particular agent cannot satisfy the task. The Functional Agent collects all the bids, analyzes the bids and determines if additional planning is needed for residual requirements (e.g., air refueling, additional support, etc.), creates sub-tasks for these requirements, sends those to the appropriate Functional Agent, collects the bids for the sub-tasks, composes the response plan, assembles the response bid, packs the bid into the response message, and sends the response to the requesting agent, in this case the AOC Node Agent.

All the planning actions implemented by the planner follow a similar workflow of messages and control flow: The process starts with the user interface sending a message to the Planner UI Server hosted in the AOC Planning Node. The Planner UI Server forwards the message to the AOC Node Agent. The AOC Node Agent decomposes the user request into tasks and assigns those tasks to Functional Agents. The messaging infrastructure is responsible for delivering the messages with the tasks to the Functional Agents. The Functional Agents then delegate the actual task processing to the Resource Agents. The plan snippets generated by Resource Agents flow back to Functional Agents, get composed into larger plans, and finally flow back to the AOC Node for final ATO generation.

The Messaging Infrastructure

The flexibility that the Distributed Control Architecture provides by hosting any agent in any node is achieved through a messaging infrastructure that mediates all interaction among agents in a location-independent fashion. It should be clear by now that a SCHARP Planning Node is just a collection of Planning Agents running in the same Java Virtual Machine. Similarly, a SCHARP Planning Cluster is a collection of interconnected Planning Nodes. The actual location of the agents is pretty much irrelevant to the operation of the system. The Messaging Infrastructure was designed to allow agents to communicate with each other uniformly, and independently of their location. All agents communicate via message passing using either the actual AMQ broker service, or some internal messaging optimization that emulates the behavior of the message broker to improve intra-node communication. There is no actual coupling among agents.

The messaging infrastructure has 4 main components:

1. The ActiveMQ message broker responsible for managing the message traffic among agents.
2. An Agent Client component (e.g., a Java class) that act as a proxy for interacting with agents.
3. An Agent Server component (e.g., a Java class) that actually implements the processing of the messages.
4. An Agent API (e.g., a Java interface) that greatly simplifies the interactions among agents by providing a small set of message processing primitives that encapsulate the details of the messaging mechanism.

The Agent API is a set of methods (e.g., a Java Interface) all agents must implement. In the previous section, we talked about agents sending messages to other agents. The actual sending and receiving of messages is handled by the methods implemented by this API. The message sending mechanism is similar to a Remote Procedure Call: when an agent S wants agent R to process some task, agent S calls the appropriate task processing method in agent R's Agent API using a proxy for agent R. The implementation of the Agent API method takes care of packing the task into an AMQ message, sending the message to the actual agent R via the AMQ, getting the response from agent R, unpacking the result, and returning the result to agent S. Agent S has no knowledge of the actual location of agent R.

The Agent Client, and the Agent Server are Java classes that implement the client side and server side respectively of the Agent API. In our example above, the proxy used by agent S to call a method on agent R is an instance of the Agent Client. The server is the actual agent: All agents are instances of the Agent Server. The Agent Server implements the control loop for an agent. The server listens to AMQ messages, and triggers agent functionality in response to received messages.

At node startup, the Node Agent creates one instance of an Agent Client for each agent that should be present in the Planning Cluster. This includes all the remote Node Agents, all Functional Agents, and all Resource Agents. Each Node Agent has clients representing proxies to all other agents in the system. This allows a Node Agent to potentially communicate with any agent. The Node Agent also creates an instance of an Agent Server for each agent hosted in its node. Each Agent Server runs as a separate thread in the node's Java virtual machine. Each Agent Server has a control loop that keeps listening to AMQ messages and dispatches those messages to the appropriate message handler. Each Agent Server has a dedicated AMQ channel (e.g., an AMQ queue or topic) for receiving messages. Both the Agent Client and the Agent Server know the unique identifier of this channel. The Client writes messages to the channel - it is a message producer, and the Server reads messages from this channel - it is a message consumer.

For each agent in the cluster, there is an Agent Client instance corresponding to that agent in every node. There is, however, only one Agent Server instance for each agent in a cluster. Only the node hosting the agent has an instance of the Agent Server. The DCP document specifies where Agent Server instances should be hosted. There is a single AMQ channel per agent. This channel has multiple message producers, and only one message consumer. The server receiving a message does not send a response to that message through the same channel from which it received the message. It always uses the dedicated agent channel. Each message contains the unique identifier of the message's sender. If there is a need to respond to a message, the Agent Server generating the response uses the Agent Client instance corresponding to the sender of the message to respond. Let's revisit our communication example in more details. Agent S requires Agent R to process some task. Each node has a directory service (e.g., a lookup table) of all planning agents in the system. Agent S uses this lookup table to get a handler to Agent R. The lookup table returns an instance of an Agent Client for Agent R. Agent S then calls the appropriate task processing method on the Agent Client object. The client creates an AMQ message for the task and sends it to the channel the corresponding Agent Server R is listening to.

Agent Server R receives the message, processes the task, and follows the same workflow to send a response back to Agent S: Agent R retrieves the identifier of the sender from the message, uses the lookup service to find a handler to agent S, receives an instance of a Client Agent for agent S, calls the appropriate method on the client object, and the client sends the message to Agent S's unique channel. Agent Server S processes the response.

If both agents are hosted in the same node, it is possible to bypass the sending and receiving of the message using the AMQ broker. When the receiver of the message is hosted in the same node as the sender, the communication infrastructure forwards the message directly to the server without going through the AMQ channel.

In earlier implementations of the system, all agents were using the AMQ broker for all messages. With this optimization there is no external network traffic when agents in the same node are communicating. This optimization greatly reduced the communication overhead, especially under restricted bandwidth conditions.

Mission Planning Algorithms

In the previous section we introduced the generic problem-solving workflow to explain the distributed control architecture, and the message passing mechanism. In this section we will describe in more detail the logic and algorithms used to plan the different types of mission.

As we saw before, the planning process starts with the AOC Node Agent sending top-level tasks, or mission requests, to the different Functional Agents. Each Functional Agent is responsible for planning only one or a few closely related types of missions. Figure 4 shows the general message flow for a strike mission: The AOC agent sends a message with a task to the Strike Functional Agent. The Strike Functional Agent then interacts with Resource Agents and other Functional Agents to compute a package that can satisfy the request.

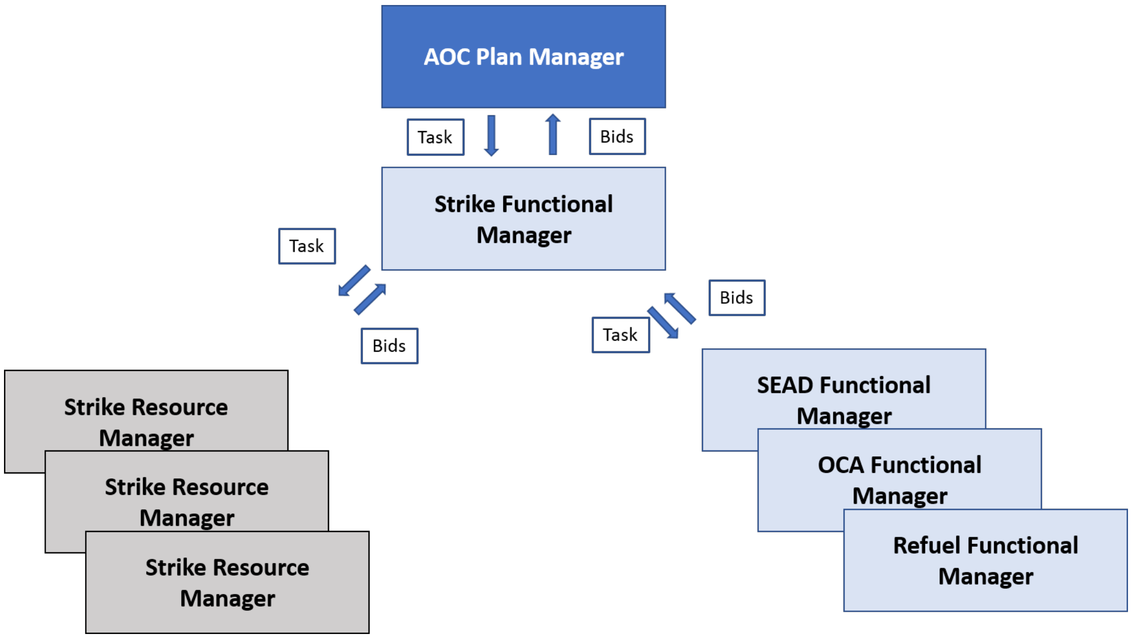

Figure 4: Problem Solving Workflow

The current implementation has the following types of Functional Agent:

- STRIKE Agent: Plans Air to Ground Strike packages and missions.
- XATK Agent: Plans Air-to-Ground Alert missions.
- KI Agent: Plans Kill-Box Interdiction Air-to-Ground Strike missions.
- C2 Agent: Plans Command and Control Orbit missions.
- DCA/OCA Agent: Plans Air-to-Air alert and escort missions.
- CAS Agent: Plans Close Air Support missions.
- SEAD/XEW Agent: Plans EW alert and escort missions.
- ISR Agent: Plans Intelligence, Surveillance and Reconnaissance missions.
- REFUEL Agent: Plans Tanker missions and Air Refueling support.

Each Functional Agent delegates the actual mission creation to the more specialized Resource Agents. The Planner's Resource Agents correspond to the actual Air Units available in the theater. Each Resource Agent has a pool of aircraft resources available for planning. The constraints on resource availability at each Air Unit are defined by the Unit Contract provided as part of the scenario input data. The Resource Agent models the GO availability profile defined in the Unit Contract. Each resource agent "knows" how to create missions of a particular type using the types and quantities of resources available.

The Resource Agent manages a homogeneous fleet composed of a number of aircraft of the same type. It creates a mission by doing the following:

1. Estimating resource requirements for the task.
2. Reserving available resources from its pool.
3. Computing detailed and optimized flight routes.
4. Scheduling start and end times for each flight segment.
5. Prescribing appropriate munitions or additional equipment to use.
6. Estimating munition usage.
7. Calculating fuel utilization.
8. Identifying the need for additional support missions.

A Resource Agent is usually dedicated to only one mission type, and usually communicates with only one Functional Agent. For example, the STRIKE Functional Agent interacts with a number of STRIKE Resource Agents. We usually refer to the Resource Agents simply as units. For example, we will refer to a STRIKE Resource Agent as a strike unit. Similarly, we refer to the Functional Agents as functional managers or just managers. We call the STRIKE Functional Agent simply the strike manager. The AOC Node Agent is called the Plan Manager.

Although there are 9 different types of Functional Managers, from a more abstract planning perspective, there are 4 distinct types of mission structure that are planned by the different functional and resource agents:

1. A strike mission or package.
2. An orbit mission.
3. A support mission.
4. An air refueling mission.

Planning Strike Packages

A strike package is a set of missions containing more than one single mission. Generally, the set comprises one or more EW Escort missions, one or more OCA escort mission, plus the Air Refueling that may be needed by missions in the package.

The planning of strike packages is the most complex planning activity performed by SCHARP. It requires the synchronization of several missions using different types of aircraft flying from different locations. Figure 5 shows a more detailed view of the workflow for planning strike packages.

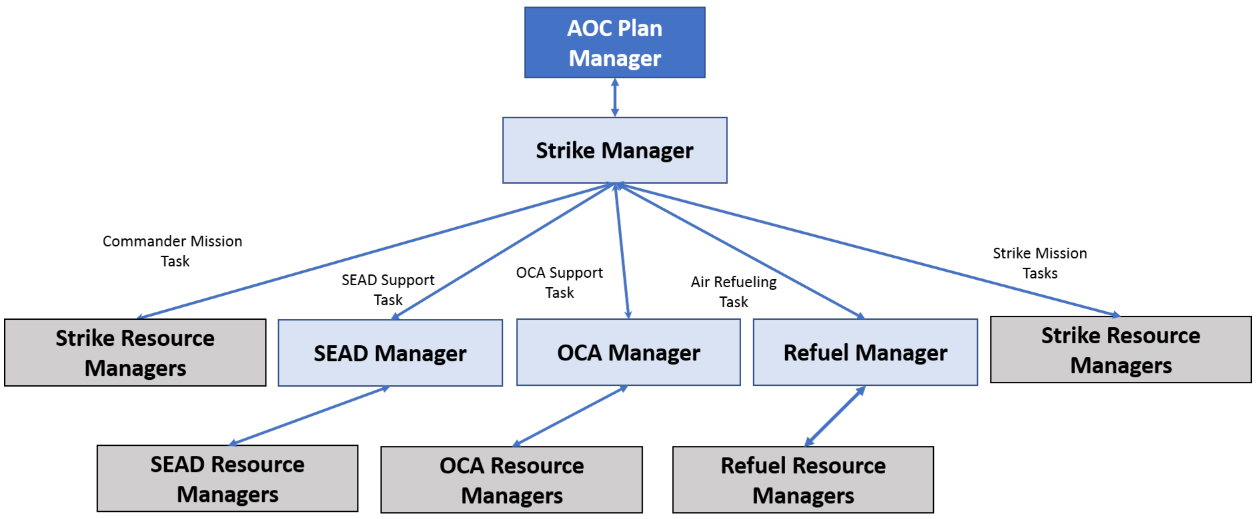

Figure 5: Strike Mission Detailed Planning Workflow

The Strike Manager is the functional manager responsible for planning strike packages. The planning of a strike package starts with the creation of a top-level strike task. The Plan Manager (e.g., the AOC Node Agent) creates top-level strike tasks corresponding to the targets in the JIPTL, and sends those tasks, one at a time, to the Strike Manager.

A strike task may contain a single target, with all its JDPIs, or multiple targets clustered together. If the strike task contains more than one target, the strike package must successfully plan missions to hit all targets in the target cluster specified by the task. The Planner does not currently provide partial satisfaction of a task. The system does not break a single target into multiple tasks. It may create multiple missions for a single target if the number of JDPIs for the target is such that a single mission cannot strike all of them.

The Strike Manager starts planning the strike package once it receives the message with the task from the Plan Manager. The strike manager uses the following workflow to create a strike package:

1. It first plans the package's commander mission: The commander mission is the first strike mission planned for the package. The route and munitions used by the commander mission are optimized for the higher priority targets in the strike task. The commander mission visits all targets in the task in priority order. It deploys its munitions on the targets in priority order. Depending on the amount of munitions available

to the mission, and on the number of targets in the task, the mission may fly over some targets but not deploy any munitions on those targets. The system later plans additional strike missions to hit those targets. If a target has too many JDPIs, it is possible to have multiple missions from the package deploying munitions over the same target.

The strike manager generates the commander mission by requesting a candidate mission from each one of the strike units (e.g., strike Resource Agents) available in the cluster. The strike manager collects the bids and selects the best commander mission based on some user-defined metric like shortest mission duration, maximum amount of damage, minimum munitions usage. Strike units that cannot generate a mission for the request, return an empty plan or an empty bid.

Once the Strike Manager selects one of the feasible missions, if any, it notifies all other units providing bids that the resources in those missions can be released - we say those bids have been rejected. If the planner cannot find a feasible commander mission, an empty plan is immediately returned to the Plan Manager, and the Plan Manager marks the strike task as infeasible.

2. Each strike unit receiving a request for a candidate strike mission creates a mission optimized for the highest priority target. The creation of the commander mission by the unit implements the following algorithm:

   (a) Given the earliest start and latest finish times defined in the strike task, the unit finds all GOs (e.g., availability intervals) that have enough available unused sorties for a strike mission. If there are no available intervals with enough capacity, the unit returns the empty bid.

   (b) For each available GO or interval, the unit computes the flight route for the mission. The flight route is represented as an ordered sequence of flight legs or activities. Each activity has an origin, a destination, an earliest start time, a latest end time, and an expected duration. Each flight segment between 2 waypoints is represented by an activity.

   (c) The route computation algorithm minimizes the total time the mission spends in the battlespace. It starts by selecting the best battlespace entry point if more than one option is available and computing the route from home base to entry point. The route is then expanded to visit the locations of all targets in the task. If, after visiting all required targets, fuel and munitions are still available, secondary or opportunistic targets may be added to the mission route. Finally, a flight from the last target to the exit point is added to the route, and the route from battlespace exit location back to home base is computed. Visits to air-refueling tracks are also inserted into the mission's positioning (e.g., from home base to entry point) and de-positioning (e.g., from exit point to home base) itineraries if the aircraft in the mission cannot fly the round-trip to base without air refueling. If no feasible route can be created for a certain go, the route computation algorithm returns an empty route.

   (d) If a feasible non-empty route exists for a given go, the unit computes the parent mission that is essentially a container for the activities in the generated route. The mission object summarizes or aggregates all the relevant data in the detailed flight route defined by the activity sequence.

   (e) The unit creates a reservation for the resources required by the mission. For example, if the mission requires 4 aircraft, and the go selected was the go starting at 0600 and finishing at 1200, the unit reduces the number of available sorties in the corresponding go by 4 units.

   (f) Based on the selected start time for the parent mission, the unit may perform temporal propagation to precisely define when targets will be hit, and when air refueling will occur. The unit will update fuel consumption values for each activity and compute the overall fuel consumption for the parent mission. It will also perform some level of threat analysis and annotate the activities that are likely to be threatened by enemy SAM sites or airfields.

   (g) If a feasible strike mission is successfully created and the resource reservation succeeds, a strike mission bid is created and returned to the strike manager. Otherwise an empty bid is returned.

3. The strike manager collects all the candidate commander missions and creates candidate packages using each of these candidate missions. To do this, it first tells all the units to cancel all the reservation made for the candidate missions. It then cycles through the candidate missions one at time and computes a package for each candidate mission. It does this by canceling and re-creating the reservations provided in the bid. The next steps in the workflow are executed for each candidate commander mission.

4. If a feasible commander mission is available, the strike manager plans EW and OCA Escort Missions if necessary: The strike manager analyzes the route of the commander mission and, if it identifies potential SAM or airfield threats, it creates SEAD and OCA Tasks, and sends those tasks to the SEAD and OCA manager respectively. The SEAD and OCA managers will then try to generate EW and OCA Escort missions to support the package. A failure to plan EW or OCA escort may cause the STRIKE manager to fail planning the strike package and return an empty plan to the Plan Manager. The user can allow a strike package without either EW or OCA support to fly by setting a planning preference. If there are no support missions, and the choice is to return the empty plan, before returning the empty plan to the Plan Manager, the Strike manager notifies the unit providing the resources for the commander mission to release its resources - the commander mission bid has been rejected.

5. If necessary, the strike manager will also plan air refueling for the commander mission. The actual planning of the refueling is delegated to the Refueling Manager. If there is a feasible commander mission that requires air refueling, the strike manager creates one or more air refueling tasks, and sends them to the Refueling Manager. If air refueling is not feasible, the Planner cancels all pending reservations, and returns the empty bid to the Plan Manager. Failing to find a tanker for air refueling when fuel is needed always causes the task to fail.

6. If there are residual targets or JDPIs that have not been hit by the commander mission, the strike manager plans additional strike missions until all targets and JDPIs have been hit. If it is not possible to find strike missions to hit all targets, the strike manager cancels all previously made reservations and returns the empty bid to the Plan Manager. In case of failure, the previously made reservations that need to be canceled include any strike mission that has not yet been rejected, plus the support missions created, plus any Air Refueling activities.

7. To compute non-commander missions for a package, the strike manager repeats the following sequence of steps until all the JDPIs have been struck:

   (a) Compute the set of remaining JDPIS that have not yet been struck by the strike package. If the set of residual JDPIs is empty, exit this loop and proceed to package creation using the selected strike missions. If the residual JDPI set is not empty, go to step b.

   (b) Create a new target task including the commander mission and the remaining set of JDPIS.

   (c) Send the new strike task to each strike unit.

   (d) Collect all candidate bids, select best bid, and cancel reservation for all other bids.

   (e) Go to step a.

   The strike units also plans non-commander missions following a workflow similar to the one described for the commander mission. The main difference is that, while creating the mission flight route, the unit only computes the positioning, and de-positioning routes, and copies the flight route inside the battlespace from the commander mission. All the missions in the strike package rendezvous at the entry point defined for the commander mission and fly together inside the battlespace. All missions also exit at the same exit point and then fly separately to their home base.

8. The strike manager will not schedule additional SEAD or OCA Escort missions for the additional strike missions created. It assumes the support missions created for the commander mission will be enough to support the package.

9. If the system was able to create a set of missions that satisfy all the requirements of the strike task, the Planner succeeded in generating a plan. The Strike Manager creates a strike package with all the missions created, creates a strike bid, and returns a successful bid to the Plan Manager.

10. The Plan Manager receives the response from the Strike Manager. If the bid is not empty, it records the strike as successful and adds the returned strike package to the collection of successful packages. Otherwise, the Plan Manager marks the strike tasks as failed.

Planning Orbit Missions

The non-strike missions in the ATO are usually missions composed of 2 or 4 aircraft that stay in orbit at certain pre-defined airspace locations for a certain periods of time (perhaps refueling at a nearby air-refueling track), before finally returning to base. The Tanker mission is a special type of Orbit Mission that stays in the same orbit for the entire duration of the mission. Figure 6 shows the workflow used to plan an orbit mission. An orbit mission may refuel several times depending on the duration of the orbit.

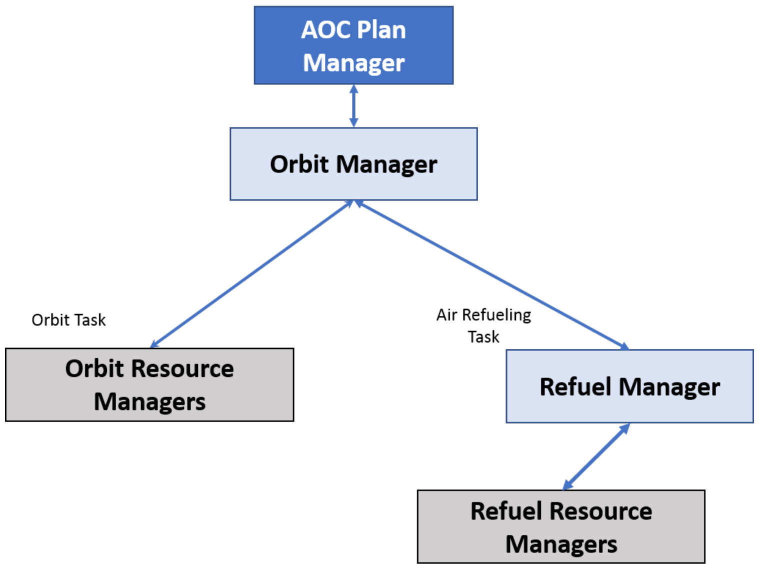

Figure 6: Orbit Mission Planning Workflow

Orbit tasks are top-level tasks created by the Plan Manager (e.g., the AOC Node Agent). Orbit tasks are defined based on input coming from the AOD, or user specified. Orbit tasks usually specify the orbit location, and the expected duration the mission should stay in orbit. Some tasks also specify the unit that should provide the resources for certain orbits.

C2 missions, CAS missions, DCA missions, XATK missions, FAC missions, and XEW missions are all examples of orbit missions. Ground alert are also treated similarly to orbit mission, but they stay on the ground instead of flying to some orbit location.

The workflow for orbit missions is straightforward:

1. The Plan Manager sends a top-level orbit task to the corresponding orbit Functional Manager.

2. If the task specifies the unit providing the resources for the mission, the orbit manager sends the task directly to the specified orbit unit. Otherwise, it requests candidate missions from all units that can plan that particular type of orbit mission.

3. If the orbit unit receiving the task does not have enough resources available in the time window specified by the task, it returns the empty bid. Tanker units, when planning tanker missions, also check if the air-refueling track specified in the task still

has orbits available. The planner assumes an air refueling track can have different orbits at different altitudes, and will assign a tanker for each orbit.

4. If the unit has enough available resources, it expands a flight route or flight plan capable of positioning the aircraft of the mission in the specified orbit. The flight route may also include as many trips to the air refueling track as necessary to keep the mission in the orbit during the requested time window. If the mission is not feasible given the aircraft or unit contract constraints, the unit returns the empty bid.

5. If the mission is feasible, and air refueling in needed, the orbit unit creates the necessary air refueling tasks, and sends the tasks to the REFUEL manager. If any of the refueling tasks fails to schedule, the orbit mission creation for that unit fails, and the orbit unit returns the empty bid.

6. If a mission is successfully created, a non-empty orbit bid is returned to the orbit manager.

7. The orbit manager usually returns the first feasible mission returned by one of the orbit units. It does not collect bids from all units.

All orbit missions, with the exception of tanker missions, are expected to take off and land at the planned time. Tanker missions can change their landing time depending on the amount of fuel offloaded. Tanker missions behave like dynamic resources: A tanker mission in a certain air refueling track tells the system there is certain amount of fuel available at that track. The initial duration of a tanker mission is the maximum time the tanker can stay in orbit given fuel and other time constraints. As the tanker offloads fuel, its available fuel, and corresponding mission end time is updated based on how long the tanker can stay in orbit given the remaining amount of fuel.

Planning Support Missions

Support missions are missions that are created to complement the capabilities of other missions planned by the system. For example, if a strike mission will fly over a potentially active SAM site, or if the strike mission is going to hit a SAM site, depending on the type of aircraft used in the mission, it may require support from an EW mission. Similarly, for airfield threats: If a strike mission will cross the radar range of an airfield, or if it is going to hit the airfield, it may need support from one or more OCA escort missions. SEAD Escort and OCA Escort are the two types of support mission the Planner currently creates to support strike missions. The planning of both types of missions follows pretty much the same workflow. Only the aircraft type and the units are different.

The planning of support missions starts with the Strike Manager analyzing a strike mission, identifying the need for some type of support, creating the support task, and sending the support task to either the SEAD Manager or to the DCA/OCA Manager.

Both managers follow the same workflow to plan support missions:

1. The SEAD or the OCA Manager requests mission candidates from each of the SEAD or OCA units available in the system.

2. If the Support unit does not have enough available sorties at the required mission time, it fails and returns the empty bid.

3. Each tasked support unit will create a mission that will follow exactly the same flight route as the strike mission while inside the battlespace. The strike mission has well defined locations, or waypoints, where it enters and exits the battlespace region. All the flight segments for the strike mission inside the battlespace are copied to the route of the support mission. The support mission is expected to rendezvous with the strike mission at the battlespace entry location and fly together with the strike mission until the strike mission exits the battlespace at the specified exit point.

4. After expanding the route inside the battlespace, the support unit computes the positioning route for the support mission from the physical location of the Air Unit's home base to the entry point location. The unit also computes the fuel consumption for the support mission and, if necessary, it adds waypoints for air refueling before the mission reaches the entry point. If the support mission cannot feasibly reach the rendezvous point in time, the planning of the mission fails, and the support unit returns an empty bid.

5. If the mission is feasible, the support unit computes the de-positioning route from battlespace exit point location to home base. The unit again computes fuel requirements and adds refueling to the de-positioning legs if necessary. If the support mission is not fuel feasible (e.g., the aircraft does not have enough fuel to fly between the times it can refuel), the unit returns the empty bid.

6. If the support mission requires air refueling, the unit creates the air refueling tasks and sends them to the REFUEL manager. If refueling is not feasible, the support unit returns an empty bid.

7. If a support mission is successfully generated, the support unit returns a support bid to the requesting functional manager (e.g., the SEAD or OCA manager). The support manager collects all the bids from its support units, selects the best non-empty bid according to some user specified criteria, and returns the best bid to the Strike Manager.

8. The Strike Manager finishes its planning by performing temporal propagation if a support mission forces a strike mission to move forward in time.

Planning Air Refueling Missions

The REFUEL Manager plans both Tanker missions and air refueling activities. Tanker missions are orbit missions planned in response to Tanker tasks. Tanker tasks are currently top-level tasks created by the Plan Manager based on estimated fuel needs, and available tanker resources and air refueling tracks. A Tanker Mission is expected to refuel multiple missions. A Tanker Mission acts as a gas station in the sky. The Planner takes care of managing the amount of fuel available in a Tanker Mission, and the amount of fuel available at an Air Refueling Track: The unit manages the fuel available in each Tanker Mission, and the REFUEL Manager manages the fuel available at refueling tracks. A refueling track may have multiple tankers coming from different air refueling units. The planner assumes each refueling track location can support up to 4 orbits, each orbit at a different altitude.

Air refueling activities are created in response to Air Refueling tasks. These activities represent the temporal and spatial synchronization of an existing Tanker Mission with some other mission. As the different planning agents create the flight route for a non-tanker mission, they identify the need for air refueling, and insert placeholders for air-refueling activities in the mission's flight path. The mission receiving fuel must fly to the air refueling track location, spend some time at that location to refuel, and then fly to the next location in its route.

If a mission requires air refueling, the agent planning the mission creates one or more Air Refueling tasks and assigns those to the REFUEL Manager. The workflow for planning of Air Refueling tasks is:

1. The REFUEL Manager receives the Air Refueling Task. The task specifies the time interval in which the refueling is required, the required Air Refueling track, and the amount of fuel required. It also specifies if the receiving aircraft needs BOOM or DROGUE refueling equipment. The REFUEL Manager selects all the Air Refueling or Tanker units that can provide the required type of refueling, sorts the units by their distance from the requested refueling track, and sends an air-refueling task to each of the air refueling units.

2. Upon receiving the task, each air refueling unit will try to find the earliest time interval in which it can satisfy the task.

3. The air refueling unit will first verify if it has at least one Tanker resource or mission in orbit at the required Air Refueling track during the time interval specified in the Air Refueling task. If no such Tanker mission has yet been created, the unit returns an empty bid. The unit will not create a new Tanker mission for an air-refueling task since it has no visibility of other Tanker missions from other air refueling units already planned to use the Air Refueling tracks.

4. If there is a Tanker mission from the unit already planned for the required Air Refueling Track during the required time interval, the unit will check if the Tanker mission has enough fuel to provide the requested amount of fuel and will check if the Tanker mission can actually refuel the requesting mission during the specified interval. It is possible that although there is a tanker from the unit in the specified track, the tanker is already allocated to refuel other missions at the specified time interval. If there is not enough fuel, or if none of the tankers from this unit on the requested track can offload fuel during the requested interval, the unit returns the empty bid.

5. If there is a feasible time interval for a tanker from this unit to satisfy the request, the Tanker unit computes the exact duration of the offload activity, computes the earliest start time and finish time for this activity, and creates an air-refueling reservation for that Tanker Mission. The air-refueling reservation keeps track of the exact start and end time of the air-refueling activity, the amount of fuel to offload, and the identifier of the fuel-receiving mission. As a result of creating the reservation, the structure of

the Tanker Mission is modified to update the remaining amount of fuel available in the tanker aircraft after the fuel offloading activity. The planner may change the end time of the Tanker Mission if the level of fuel in the tanker is too low after the end of the newly created refueling activity.

6. If an air-refueling reservation or activity is successfully created, the unit creates an air-refuel bid specifying the Tanker Mission providing the fuel, and the exact start and end times for the air refueling activity.

7. The REFUEL Manager collects air-refueling bids from all available Air Refueling units and selects the best bid. The selected bid is usually the one that can closely match the earliest start time of the request. As an optimization, while collecting the bids, if one of the units generates a bid that exactly matches the earliest start time of the request, the REFUEL Manager immediately accepts that bid, rejects any previously generated bids, and returns the selected bid to the requesting agent.

8. Upon receiving the air-refuel bid, the agent planning the mission requesting air refueling may be required to perform some temporal propagation to adjust the mission's times to accommodate the air refueling. It is possible that the air refueling times would force the mission's start and end time to shift forward in time.

9. If the REFUEL Manager returns an empty bid, the agent planning the requesting mission may decide to fail the planning of the mission, or may iterate over the available air refueling tracks, and repeat the process until it finds feasible air refueling.

Commander's Guidance

SCHARP's Commander's Guidance allows the operator to vary high-level guidance for the planner and consequently receive a range of plans with their corresponding metrics ('baseball cards').

Table 1: Example allocation of resources to tasks.

Task	Percentage of Resources to Be Allocated
AI/KI	30%
CAS	20%
SEAD	20%
OCA	15%
DCA	15%

The general procedure is:

- The user interface (UI) sends the planner parameters for allocating units' resources to various tasks. Table 1 shows an example in which 30% of the overall resources are to be used for strike (AI or KI), etc.

- The UI sends the planner a configuration for multi-role platforms, giving, for each platform, a prioritized list of roles the platform may perform. For example: the fighter planes in a given strike unit may be configured for strike first, then CAS, then DCA and finally OCA, while the fighter planes from a different unit may be configured for DCA first, then OCA and finally strike.
- The UI sends the planner no-earlier-than (NET) and no-later-than (NLT) times for targets. The UI uses this mechanism to define the flow of the day's ATO; for example, northern targets in the morning, followed by central targets in the afternoon, followed by southern targets in the evening. (By default, the targets have NETs/NLTs determined by the scenario data.)
- The UI then instructs the planner to generate a plan. The planner uses the allocations and the configuration of multi-role aircraft to generate new unit contracts for the units. It then generates a plan using those unit contracts and respecting the target NETs and NLTs.
- The UI then requests the metrics computed for the generated plan.
- This process can be repeated with different allocations and NETs/NLTs and the metrics compared; e.g., a DCA-heavy plan may be compared with an OCA-heavy plan, or a north-south flow with an east-west flow.

The Allocation Algorithm

The inputs from the user interface to the allocation algorithm are the allocation parameters (i.e., a percentage of the total resources to be allocated to each task) and the preferred roles for multi-role aircraft. The algorithm also uses the default unit contracts to determine how many sorties each unit can provide in each go. The output is a new set of unit contracts allocating those sorties to the various tasks.

The algorithm works as follows:

1. The unit contract for a specific go for a specific unit are partitioned into roles: the number of sorties for air-to-air, the number of sorties for strike, the number of sorties for SEAD, etc. The allocation algorithm sums over the roles, to determine the total number of sorties available from that go.
2. The algorithm sums the sorties over all goes, to determine the total number of sorties in the ATO day.
3. The algorithm then uses the allocation percentages to determine how many sorties should be allocated to each role: sorties for strike = total sorties $\times$ percentage for strike, etc.
4. The algorithm then tries to create unit contracts in which the total sorties for each role match, as closely as possible, the number of sorties determined for that role. (It may not be possible to achieve this completely.)

- The algorithm starts with the strike role.
- It iterates through the units. If the unit's platform has strike as its first preferred role, then the algorithm allocates the unit's sorties to strike.
- It continues this until either enough strike sorties have been allocated, or all units have been tried and not enough strike sorties have been allocated. In the latter case, the algorithm goes through the units again, this time allocating sorties from units whose second preferred role is strike. And so on. Eventually, either enough strike sorties are allocated, or all units and preferred roles have been exhausted.
- The algorithm then repeats for the CAS role, allocating sorties than were not previously allocated to strike. And so on through each role.

Conclusions

This report documents the architecture and algorithms of SCHARP's planning engine.

The architecture is flexible, allowing fully centralized operation (with all planning for all resources performed on a single node), fully distributed operation (with each air unit and functional manager operating on its own node), and partially distributed operation.

The planning algorithms follow a hierarchical decomposition, with planning for the JIPTL targets and various orbit tasks at the top, decomposed into planning for functional areas (e.g., strike, SEAD, OCA, and refueling), which in turn decompose into planning for specific air units.

The planning algorithms use a mixture of availability profiles (that reflect the number of resources allocated by units over time) and constraint propagation to determine feasible, synchronized operations between the various resources: e.g, SEAD and OCA missions escorting strike mission through enemy airspace, and tanker missions providing fuel to other missions.