2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
AUTONOMY, ARTIFICIAL INTELLIGENCE, AND ROBOTICS TECHNICAL SESSION
AUGUST 11-13, 2020 - NOVI, MICHIGAN

An Open, ROS2, AGVRA-Based Autonomy Software Architecture for Military Robotic and Autonomous Systems

William Thomasmeyer[1], Jon St. John[1], Dave Martin[2], Rich Mattes[2]

[1]National Advanced Mobility Consortium, Ann Arbor, MI
[2]Neya Systems, Pittsburgh, PA

ABSTRACT

This paper reports on a prototype project to develop and mature a common, open, comprehensive, collaboratively developed, Government-owned, autonomy software architecture for ground robotic and autonomous systems (RAS). The prototype architecture, codenamed ***"SCION"*** *promises the flexibility needed by the both the Government and industry research, development, testing, and engineering (RDT&E) communities to leverage reusable software and more rapidly innovate new capabilities; while ensuring the discipline and enabling the modularity required to develop RDT&E software structured to meet the software safety, cybersecurity, upgradeability, and other needs of RAS programs of record. Accordingly, program offices can adopt an acquisition strategy that requires compatibility with the de facto, military RAS standard, SCION architecture, while providing OEMs with baseline, SCION-compatible, GFE software (e.g. a future version of RTK). Such a strategy encourages optimal innovation, increases competition, provides for greater IP flexibility, and incentivizes OEMs to propose solutions based on the GFE software, while ensuring that the responsibility and accountability for the software remain with the OEM.*

Citation: A. William Thomasmeyer, Jon St. John, Dave Martin, Rich Mattes, "An Open, ROS2, AGVRA-Based Autonomy Software Architecture for Military Robotic and Autonomous Systems", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 11-13, 2020.

1. INTRODUCTION

As the United States ARMY CCDC Ground Vehicle Systems Center (GVSC) endeavors to achieve the ARMY Robotics and Autonomous Systems (RAS) strategic objectives, multiple simultaneous programs have emerged, each with a different purpose, scope and timeframe. Programs and projects related to offering effective autonomous robotics solutions to the warfighter include:

- **Robotic Technology Kernel (RTK)**
- **Autonomous Ground Re-supply (AGR)**
- **Combat Vehicle Robotics (CoVeR)**
- **Modular Autonomy and Robotic Software (MARS)**
- **Robotic Operating System – Military (ROS-M)**
- **Autonomous Ground Vehicle Reference Architecture (AGVRA)**

The varying developmental timeframes and scope of each of the programs mentioned above poses unique challenges, including those related to differences and deviations from program to program in the software architectures. At the same time, a growing topic of discussion among the many Army organizations responsible for the development, fielding, and maintenance of unmanned ground vehicles is the need for an open, common, and modular software architecture (ideally aligned with the emerging AGVRA framework) that would accelerate the development and eventual fielding of autonomous tactical behaviors for military RAS.

As part of their next phase plans, several of the above named projects are beginning to take preliminary steps to reconcile the differences in the current software architectures. For example, the plans for AGR increment 3 include an effort to merge elements of the RTK architecture into the existing ROS1 based, AGR architecture. And the MARS project proposes in year 2 to undertake a study to identify one or more candidate next generation architectures for the RTK code base for a possible migration effort in FY22. To date, however, there has been no organized effort focused on working with Government and Industry stakeholders, including those involved with each of the above projects, on the longer term goal of defining a unified, open, common, next generation, autonomy software architecture for military RAS.

Under the AGVRA task request, however, the National Advanced Mobility Consortium (NAMC) has completed Phase 1 of an effort that might prove to be a significant first step in the direction of unifying the two current, primary ground architectures (RTK and AGR) and providing the multiple, interrelated GVSC efforts described above with a common next generation architecture that they can migrate towards in accordance with each initiative's own goals, objectives, milestones, and timelines.

2. Scope

The NAMC was tasked with defining a plan to generate a prototype target system architecture (TSA) as a test case for the recently released Autonomous Ground Vehicle Reference Architecture (AGVRA) V1.0 framework. The AGVRA framework provides guidelines, best practices, and model-based architectural resources informed by various systems in the autonomous ground vehicle domain. These artifacts, as laid out in the AGVRA Concept Description are intended to guide the development and implementation of ground vehicle system architectures from both a technical and a business practice perspective.

The NAMC formed a team of architecture subject matter experts from among its members for the purpose of defining a ground vehicle TSA, codenamed "**SCION**", guided by the recommendations in the AGVRA Concept Description and AGVRA Version 1 Work Products. The NAMC team set out to define the SCION architecture for a selected robotic system and representative operational mission in alignment with AGVRA principles. Driven to a significant extent by factors critical to the Robotic Combat Vehicle (RCV) program, the robot selected to serve as the target system was a RCV(Light) surrogate vehicle and the representative mission was route reconnaissance. Also consistent with AGVRA guidelines, the process of defining the prototype TSA involved analyzing existing autonomy software architectures and considering how their differences might be reconciled.

Initially, the primary purpose was to create an architecture design based on the AGVRA principles so as to provide effective and practical feedback to the AGVRA team, including suggested improvements and recommendations for improving the usefulness of AGVRA artifacts for industry and Government stakeholders. As work progressed, however, it became increasingly clear that the draft SCION TSA in fact represented an early prototype of a much broader, unified autonomy software architecture applicable to a wide range of unmanned ground vehicles. One that, with additional effort, might be readily matured into a comprehensive, common, Government-owned open, autonomy software architecture for military RAS that is:

- Based on existing, open standards like IOP, common middleware including ROS2, and an open data/information architecture (UCS)
- Grounded in and part of the AGVRA ecosystem
- Derived from merging the two proven ground robotics autonomy software architectures (RTK and AGR)

Such an autonomy software architecture would accelerate alignment to the overall ARMY modular open systems approach (MOSA) by: (i) providing a transition path in the form of an extended/enhanced common architecture that each program can leverage as their local designs progress; and (ii) delivering a *de facto* military RAS standard that industry and Government alike can use to build their future programs of record, IRAD efforts, and other plans around (similar to the effect that IOP has had).

3. Autonomous Capability Requirements

Using the RCV(L) platform and route reconnaissance mission as a basis, a series of high-level ground vehicle autonomy and related requirements were derived to guide the architecture design.

Autonomy Requirements

AUT-1: The platform must autonomously navigate to a given global waypoint.
The resupply mission requires navigation to one or more global waypoints along the path to its operator provided goal position. The platform's autonomy must provide an autonomous waypoint navigation function, which uses the platform's current position and waypoint goal to plan and execute motion to a globally referenced waypoint point.

AUT-2: The platform must detect and avoid obstacles along its path of travel.
As the vehicle makes its way to a goal position, it must consistently be planning a path through traversable space towards the destination. As part of the planning process, the vehicle must consider perceived information about the environment to avoid getting stuck among the terrain or damaging the platform.

AUT-3: The platform must accept and act upon information from external sources.
Inputs provided to the platform by an operator, such as keep-out areas, waypoints, stealth areas, etc. must be considered by the autonomy as the mission is executed. In addition, behaviors like formation following require coordination with other manned or unmanned assets. These parameters should be incorporated into the decision making of the autonomous behaviors, and possibly in the decision making as to which behaviors to execute at a given point in the mission.

AUT-4: The platform must provide state and mission feedback to external entities.
When an operator is interacting with the platform, the platform must provide feedback to the operator as to the system health and state, and the parameters and progress of its commanded mission. The platform must also be able to coordinate and share information with other manned or unmanned assets.

AUT-5: The platform must implement safety measures when operating autonomously.
When the platform is operating in an autonomous mode, it must implement safety measures such as providing visual feedback as a warning to the operator or any other bystanders. This visual feedback is required for test and evaluation and may be disabled as needed during field and tactical operations.

AUT-6: If the platform determines it is unable to complete its mission, it must execute a contingency plan.
As the vehicle attempts to navigate to a new goal position, it may encounter a situation where it is unable to reach its goal destination. In that case, the vehicle must fall back to a contingency plan to allow for recovery.

AUT-7: The platform must autonomously retain and navigate a previously traversed path.
In situations where the vehicle reaches a dead end or must otherwise turn back from its planned course, it should use previously executed motion to inform retro-traversal. If the vehicle can record and re-trace its path, it can leverage the previous efforts to detect and classify objects in the environment and find a navigable path to the goal point.

Perception Requirements

PER-1: The platform must be able to determine whether terrain is traversable.
Autonomous navigation through unstructured terrain given in AUT-2 requires perception capabilities to determine whether terrain is safe to traverse or not. The platform requires the capability to process and combine sensor streams and any a priori information to determine the best path to navigate to its destination.

PER-2: The platform must maintain an estimate of its pose within the environment throughout the mission.
Navigation to a globally referenced waypoint per AUT-1 requires that the platform must be able to track its own motion through the environment and provide an estimate of

its pose in a global reference frame. The pose information can also be used by other components, such as a world modeling component to track the locations of obstacles as the vehicle moves.

Sensing & Communication Requirements

SEN-1: The platform must support sensing the environment around it.

The platform must be able to sense the environment around it to perform autonomous navigation. This includes being able to sense the ground, and any environmental features that may make terrain unsafe to navigate or traverse.

SEN-2: The platform must support sensing its position in the environment.

The platform's sensing must be able to support localizing the vehicle within its environment, and within a global reference frame.

SEN-3: The platform must communicate with an external controller.

A route reconnaissance mission requires mission parameters such as the formation members, a goal point, any keep-out areas in the operating area, etc. The platform may also be able to provide feedback about things like its system health, current operating state, and available autonomous missions/behaviors. The platform must support a communication channel to an operator such that the mission parameters and execution can be specified by the operator, and platform feedback can be provided to the operator.

Actuation Requirements

ACT-1: The platform must provide actuation to support controlling the motion of the vehicle.

An unmanned platform has several degrees of freedom in actuation, including at a minimum steering, throttle, braking. Platforms may also include additional control such as transmission gear selection, powerplant enablement, differential locking, etc. These degrees of freedom must be actuated and exposed to the autonomy such that the desired trajectories can be translated into actuation commands for the platform.

ACT-2: The platform must provide actuation to support safe operation around humans.

In addition to the motion of the vehicle, the platform may support additional actuation such as warning lights, audible warnings, and motion of additional modular payloads. These actuation modes must be exposed to and controllable by the platform's autonomy.

4. Architecture Requirements

Using the autonomous capability requirements described in Section 3, the architectural requirements listed in Table 1 can be derived.

Table 1. Architectural Requirements Summary

Ref	Requirement	Derived From
ARCH-1	The architecture must provide platform localization in a global frame.	*PER-2*
ARCH-2	The architecture must provide communication paths to one or more sensors to detect features about its position and the environment.	*SEN-1*
ARCH-3	The architecture must provide a common communication framework to support communication with external systems (OCU, other vehicles, etc.)	*SEN-3*
ARCH-4	The architecture must provide a common communication framework for passing data between components internal to the architecture.	*PER-1, PER-2*
ARCH-5	The architecture must provide a consistent time base for data synchronization between multiple sensors, systems, and external communications.	*SEN-1, SEN-2*
ARCH-6	The architecture must provide a perception system to process sensor data from one or more sensors and classify relevant features in the environment	*PER-1*
ARCH-7	The architecture must provide support for commanding the motion of the vehicle.	*ACT-1*
ARCH-8	The architecture must provide support for commanding visual indicators and other safety actuation.	*ACT-2*
ARCH-9	The architecture must provide a framework for arbitrating between multiple autonomous behaviors that may run in parallel.	*AUT-1*
ARCH-10	The architecture must continuously monitor the health of all subsystems and take appropriate action in the case of the failure of a subsystem.	*AUT-5*

ARCH-1 Global Localization

The platform is required to provide a consistent pose (position, orientation, velocity) estimate in a globally referenced (e.g. latitude/longitude) frame. This pose estimate is required for functions such as navigating to a fixed point, geo-referencing sensor data, position coordination between multiple platforms, and closed-loop control of speed and steering.

ARCH-2 Sensor Support

To complete the desired mission, the platform is likely to need input from many sensors, such as actuator feedback, LIDAR, camera, odometry, GPS, etc. The architecture must provide enough connectivity and bandwidth to allow all required sensors to be read and processed at their natural rate. This may include support for Ethernet, Serial, CAN, or other custom Inputs and Outputs (I/O) as needed.

ARCH-3 External Communication

Parts of the mission require accepting input from and providing output to an external system, such as an OCU. The architecture should provide a communication framework and interface to communicate with external systems. Per the AGVRA Concept Description, the chosen framework should be an industry standard protocol to enable interoperability with new and existing external systems.

ARCH-4 Internal Communication

The internal components of the architecture will need to communicate with each other to pass sensor data, commands, perception results, current positions, etc. The internal communication between components in the system shall be consistent and well-defined. Per the AGVRA Concept Description, the chose framework should be an industry standard or widely used framework to enable interoperability and modularity for components within the system.

ARCH-5 Consistent Timebase

As data needs to be associated between more than one acquisition source, and potentially more than one platform, a consistent timebase is needed to accurately time-stamp data. This includes samples from sensors, and data from external sources.

ARCH-6 Perception Subsystem

The architecture should implement a perception system that provides up-to-date information about the latest perceived state of the environment for autonomous decision making. This perception framework should handle various sensor input streams and identify features required for autonomous behaviors such as terrain traversability, safe and unsafe areas of travel, etc.

ARCH-7 Vehicle Motion

The platform must be controlled by the autonomy to traverse through difficult to navigate terrain. The architecture must provide an interface to the actuation on the platform to allow the autonomy to execute this low-level vehicle control.

ARCH-8 Safety Indicators

When the platform is operating autonomously, it must indicate it is in an autonomous mode in a way that operators and bystanders in the immediate area can use to stay clear of the vehicle. The architecture must provide a way to expose this type of indication to the autonomy.

ARCH-9 Autonomous Behaviors

Due to the diverse needs of the mission profile, multiple independent behaviors will be required to accomplish individual autonomous tasks (e.g. plan to waypoint, maintain a formation, retro-traverse a previously traveled path, hold at a goal point, etc.) These behaviors should be isolated and implemented separately to support re-use of the behaviors in other mission profiles, and extension of the platform for additional missions. The architecture must provide a way to execute and arbitrate between multiple behaviors running in parallel.

ARCH-10 System Health

To ensure safe operation, the architecture must monitor the critical components of the system, such as motion execution, sensor health, connectivity between components, etc. When a failure occurs in a component of the system, the architecture must take appropriate action depending on the impact of the failure on the overall mission. This may include disabling a sensor from the perception pipeline if it is found to be faulty, coming to a stop if the vehicle is no longer able to steer or detect obstacles, or retro-traversing the previously traversed path until the error can be reported to an operator.

5. Prototype SCION Architecture

The existing architectures and standards used by GVSC and outlined by AGVRA provide an excellent baseline for a next generation, autonomy software architecture that meets the behavior requirements outlined in Section 3. The prototype SCION architecture, described below, leverages those offerings and extends on-going efforts endorsed by GVSC into a single, unified design that pulls from the best-in-breed of existing architectures, including Autonomous Ground Resupply (AGR) and the Robotic Technology Kernel (RTK), and proven communications protocols, including IOP/JAUS, UCS, DDS, and ROS2.

5.1 System Decomposition

The overall system was first decomposed into constituent elements to specify the scope and boundaries of the autonomy software architecture. This modular approach also promotes re-use of existing system design approaches, such as the IOP Instantiation mechanism which focuses on standards-compliant interfaces, while emphasizing platform components that need additional specification not generally covered by the IOP.

Error! Reference source not found. shows a r epresentative system decomposition into a user interface, high-level vehicle control, and low-level hardware control layer taken from the Autonomous Ground Resupply program.

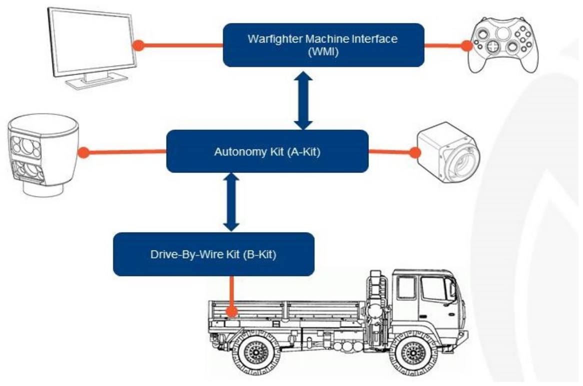


Figure 1. High Level Architecture Decomposition

- The **Warfighter Machine Interface (WMI)** is a particular application used across a number of US Army CCDC Ground Vehicle Systems Center (GVSC) programs. Generally, however, this top level component is meant to represent a user interface by which the warfighter or other operator interacts with the system.
- The **Autonomy Kit (A-Kit)** represents the "brain" of the platform that enables the intelligent behaviors identified in Section 3. The A-Kit generally focuses on platform mobility, with emphasis on path planners, obstacle detection/obstacle avoidance (ODOA), localization, etc.
- The **Drive-By-Wire Kit (B-Kit)** encapsulates the low level interfaces into the hardware of the platform. Because these interfaces can often be proprietary or include proprietary extensions to common vehicle standards like CAN or J1939, the B-Kit abstracts away some of those particulars, offering a common and consistent interface regardless of underlying platform.

The goal of the prototype SCION architecture is to define the interfaces to and from the A-Kit, while also decomposing the A-Kit into notional constituent components and interfaces.

Figure 2 shows the use of IOP between the different layers of the architecture. This diagram also proposes the use of IOP (and by extension, JAUS) for payloads such as manipulators, pan/tilt systems, UAVs, and other external entities. Sensors internal to the A-Kit, such as cameras, GPS, and LIDARs are intentionally absent from this decomposition. Because of the need for tight integration with the world model and the rest of the A-Kit, these interfaces are considered part of the A-Kit and will be described in the next section. Sensors external to the A-Kit, such as those provided by payloads, are expected to be managed by the payload module and made available to the A-Kit via IOP.

Note that the external client listed in Figure 2 shows a generic "controller", generally taken to mean any user interface that a human operator interacts with. However, this is not limited to traditional one-controller/one-platform interfaces like the WMI. By using interoperable protocols like the IOP, more complex external clients, such as but not limited to multi-agent planning systems and AI-based scheduling systems, are supported natively.

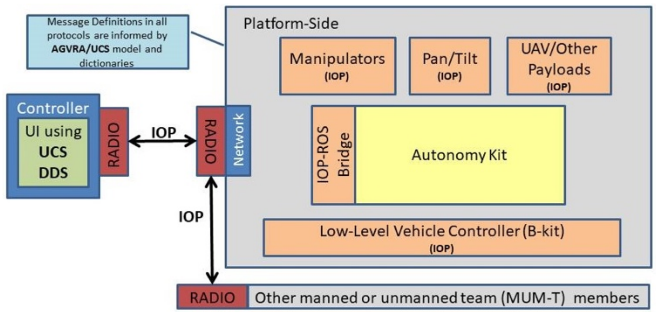

Message Definitions in all protocols are informed by AGVRA/UCS model and dictionaries
Platform-Side
Manipulators (IOP)
Pan/Tilt (IOP)
UAV/Other Payloads (IOP)
Controller
UI using UCS DDS
RADIO
IOP
RADIO
Network
IOP-ROS Bridge
Autonomy Kit
IOP
Low-Level Vehicle Controller (B-kit) (IOP)
RADIO
Other manned or unmanned team (MUM-T) members

Figure 2. Top Level System Decomposition

5.2 A-Kit Decomposition

Based on the architecture requirements defined in Section 4, the A-Kit is expected to support several top- level elements that provide specific capabilities. Figure 3 shows an initial decomposition into twelve modules.

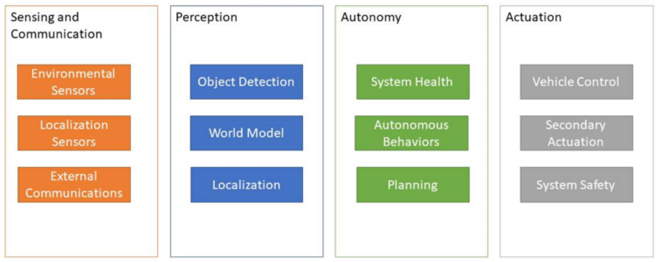


Figure 3. Initial A-Kit Decomposition

The intended overall functionality of each component is as follows:

- Sensors: Interfaces with sensing hardware, such as cameras, LIDARs, GPS, etc.
- Object Detection: Extracts features and other information from the raw sensor data
- Localization: Provides a representation of the platform location by fusing data from multiple sensors, such as GPS, INS, odometers, etc.
- World Model: Uses perception and localization to convert sensor data to a representation of the environment, often using cost maps or similar representations.
- Autonomous Behaviors: Manages the general state and operating mode of the A-Kit, choosing between multiple and potential competing objectives.
- Planning: Provides vehicle motion planners based on the current objectives.
- Motion Executor: Interfaces with the B-Kit to provide low-level motion control.
- Comms Module: Where necessary, bridges between internal and external protocols used by the A-Kit.
- System Health: Provides a system monitor that measuring the current state and health of the system, providing feedback to the human operator and other components within the A-Kit.
- Safety Checker: Monitors and approves the commands generated by the planners to ensure safe operation of the platform.

The notional decomposition proposed in Figure 3 follows the 4D/RCS design paradigm of sense-interpret-act and demonstrates strong overlap with the AGR and RTK architectures. The overall goal is to unify terminology across the two programs into a single top-level architectural representation, based on lessons-learned while following guidance provided by AGVRA. Subsequent sections provide further detail of each component and sub-component in terms of expected function within the system, the high-level data items that flow between those elements, and specify the message definitions that comprise the inputs and outputs.

5.3 Architecture Components

Figure 4 provides an overview of the proposed, top-level, A-Kit, SCION architecture. Note that the arrows represent messaging are intended to show only key data exchanges for simplicity. In a complete implementation, the exchange of information is expected to be significantly more complex between all components.

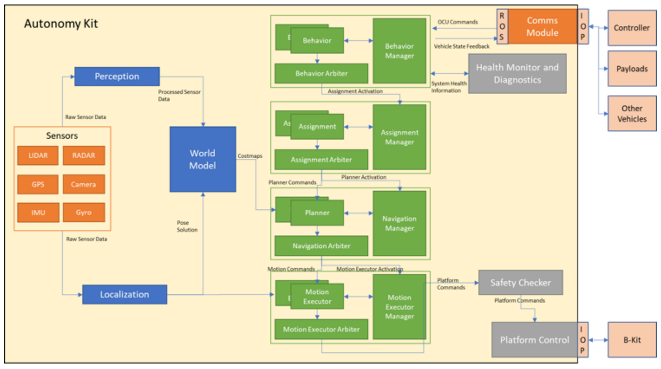
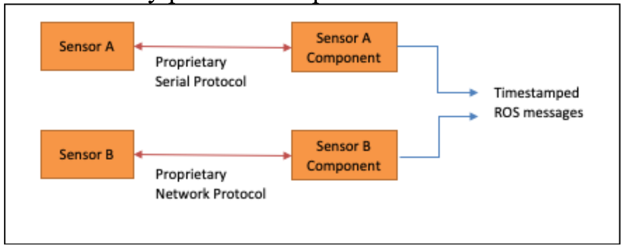


Figure 4. Proposed new architecture, high-level diagram

Sensor Components

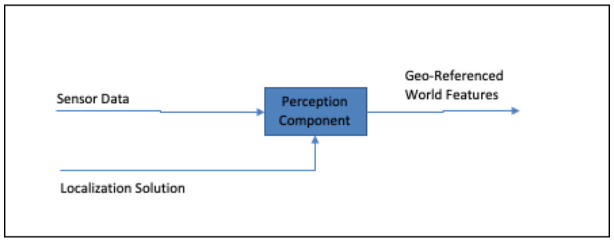
Autonomy systems generally use a suite of sensor technologies, including LIDAR, cameras (2D and 3D), GPS, IMU, and radar. These devices frequently have their own unique protocols for communication and control. The goal of Sensor Components is to abstract these unique needs into a common set of messages and a consistent representation of the output data. In a sense, the Sensor Components act as a device driver, translating between the needs of a particular sensor and instead exposing a common interface to the rest of the autonomy stack. Note, however, that the output data between sensor types can vary widely; the data representation of a 2D camera has different messaging needs than the output of a point cloud from a LIDAR. Therefore, the architecture must support several different output types based on the sensor technology selected for any particular implementation.



Note that all sensor data must be timestamped to prevent relying on old or stale data. Further, the timestamp must be common across all computing platforms within the A-Kit, using a synchronized timebase. Implementations are encouraged to use existing clock synchronization approaches such as an NTP time server or similar.

Perception Components

Perception potentially combines the data from multiple Sensor Components along with vehicle position and orientation information from Localization into a common understanding of the environment. This may include labeling data, extracting features such as traversable or non-traversable obstacles, or identifying tracks for vehicles and pedestrians. Further, Perception may be required to transform the raw sensor data from Sensor Components into a common coordinate frame, using either a global, relative, or vehicle-centric frame of reference.

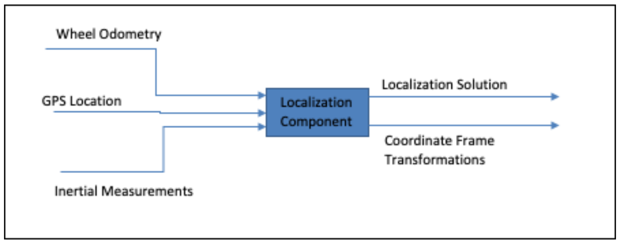


Note that, like Sensing Components, one or more Perception Components may exist within an implementation. These Components may vary in terms of their capabilities. For example, one Perception Component may be able to extract human figures and provide predictions as to future positions in space based on 2D camera imagery, while another might use LIDAR point clouds to identify the ground plane and slope of local terrain. The architecture is capable of supporting these disparate functionalities so long as each uses the common set of specified input and output messages.

Perception Components provide additional processing and analysis on the raw data produced by Sensing Components before being consumed by the World Model. As with the sensor data, the specific nature of the messaging inputs and outputs is highly dependent on the sensor modalities and the nature of the feature extraction.

Localization Component

The Localization Component is responsible for estimating and representing the platforms position, orientation, velocity and acceleration in three-dimensional space. Generally, some combination of GPS, IMU, and wheel encoders is used, but additional sensing methods are possible and must not be limited by the architecture. As a result, the inputs to the Localization Component are not defined. The implementation might use Sensor Components that wrap hardware devices or communicate with those devices using their native protocols.



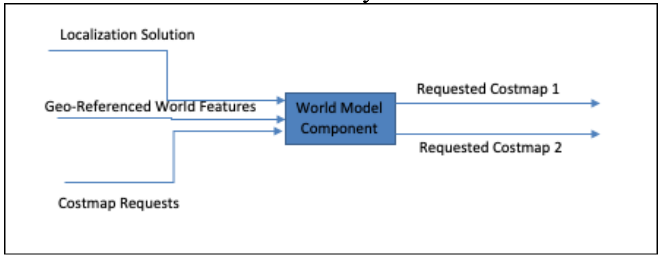
Generally, a global frame of reference, such as lat-lon, UTM, or ECEF using a WGS84 datum, is the most portable between vehicles and the OCU. However, these global frames are generally not useful for path planning as they do not support a continuous flat plane.

To that end, the localization must use a minimum of three frames: 1) a global frame completely unique across the surface of the Earth, such as lat-long, ECEF, or UTM with zone information; 2) a relative Cartesian frame that is continuous and suitable for computing the relative position of objects in the vicinity (generally under 20 kilometers); and 3) a vehicle-centric frame suitable for vehicle-relative information such as velocity and acceleration. The relationships between these three frames are tracked over time, such that it is always possible to transform coordinates between all three frames.

World Model Component

The World Model Component converts data from Perception and Localization into cost maps for consumption by Planner Components. Generally, an implementation is expected to have only a single World Model Component, which may combine data from multiple Perception Components and serve multiple Planner and Autonomy Components.

The goal of World Model is very similar to that of Perception: to interpret and provide information about the environment around the vehicle. However, while Perception is generally based on information only from current time, e.g. the most recent sensor data, the World Model maintains some history over time. This allows the World Model to "remember" obstacles that have gone out of the sensor field of view, as well as provide information about the confidence level of any obstacles detected.

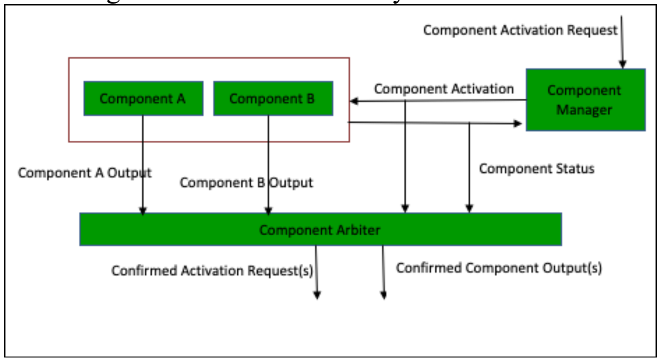


The World Model uses inputs from the Localization and Perception Components to form an understanding of the world, while taking requests for specific areas and resolutions from costmap clients. Note that the World Model must support multiple clients simultaneously, potentially providing different maps to different clients based on the topic name. Further, each client can configure multiple cost maps requests, varying in size, resolution, or frequency.

Since one-and-only-one World Model Component is expected to exist within the A-Kit, the Component uses a ROS "service" approach based on a request/response paradigm. The World Model client initiates the transaction by sending a CostMapRequest with applicable data such as size and resolution, while the World Model responds using the same CostMapRequest structure but populating the "topic" field. The client can then subscribe to that topic to receive the CostMap messages.

Autonomy Management and Arbitration

The Behavior, Assignment, Planner, and Motion execution systems each employ the same Manager/Arbiter framework, largely derived from the AGR architecture. At each level, an overall Manager controls the activation of one or more individual components, based on commands from a higher level in the autonomy stack.



At each layer, individual components register themselves at runtime with the component manager by publishing a status message to a pre-defined topic. This status message contains information about the component such as its name, a unique identifier, and feedback as to whether it is activated or has completed its task. The component manager uses these status messages to monitor the health of individual components. If a component does not respond, or fails to reflect the commanded activation state, the component manager can take action to disable or report the component.

The component arbiter is responsible for validating the output of one or more components based on the component manager's activation status, and individual component status. These validated outputs are passed to the next layer in the autonomy stack.

The Autonomy is broken up into four layers, or systems, each implementing the same Management/Arbiter framework. First is the Behavior Management System, which is responsible for setting the high-level state of the system. The next is the Assignment Management system, which supports execution of one or more autonomous tasks in support of the commanded behavior. Following that is the Planner Management system, which accepts path or waypoint commands from Assignments and generates low level motion primitives. Finally, the Motion Management Subsystem accepts motion primitives from a planner and translates them to low-level control messages for direct execution by the vehicle.

This layered structure is based on the 4D/RCS approach of multi-tiered autonomy and increasing resolution and allows for each tier to operate based on specific constraints. For example, the output of the Motion Executor Components is often published at a faster rate than the Planner Components to support low-level controller feedback and watchdog functions. Additionally, the run-time nature of component discovery and activation allows for new components to be added to the system at multiple layers without the need to change code in the manager and arbiter components.

Behavior Management System

The Behavior Management System represents the top-level autonomy of the A-Kit, managing the overall state and behavior (autonomous or otherwise) of the system. This component receives state change requests from the OCU and selects the active behavior based on the registered Behaviors, as well as the current health of the system generated by the Health Monitor and Diagnostics Component. It is responsible for safely transitioning between behaviors, and for arbitrating the output of behaviors. It is made up of three distinct components: the Behavior Manager, the Behavior Arbiter, and a collection of one or more behavior modules.

The Behavior Manager accepts inputs from external sources, such as commands from an operator, to manage the top-level state of the system e.g. teleoperation versus waypoint following versus leader/follower. Each top-level state corresponds to a behavior module. When a top-level state is commanded, the behavior manager evaluates the system state and diagnostics information from the health monitor component to verify that the interlocks are met to activate the behavior and change the top-level state. When the interlocks are successfully met, the Behavior Manager sends an activation command to the behavior.

The Behavior Arbiter acts as an intermediary between individual behaviors and the downstream assignment manager. The Behavior Arbiter receives behavior activation information from the Behavior Manager, and assignment activation requests from all running behaviors. The output from the activated behavior is re-published on a known topic to the Assignment Manager.

Individual behaviors all implement the same basic interfaces for integration into the Behavior Management System. They each publish a status message at a fixed rate which acts both as a watchdog and as feedback for the Behavior Manager as to the operational state of the behavior. Behaviors all subscribe to the Behavior Manager's activation topic and activate themselves when indicated. Finally, behaviors publish activation requests for Assignments that implement the behavior. These commands are verified and re-published by the Behavior Arbiter. Multiple behaviors may register with the Behavior Management System, but only one behavior can be active at a time.

Although the Scion architecture defines the coordination within the Behavior Management System, it is not intended to restrict the capabilities of any individual behavior module. This modular approach allows for a wide range of behaviors from simple teleoperation to bleeding-edge autonomous agents and AI-based intelligent reasoners. This flexibility and future growth is one of the core benefits of the modular, layered design.

Assignment Management System

The Assignment Management system controls the activation, execution, and arbitration of outputs from individual Assignment modules. Assignments are components that implement autonomous functions in support of a high-level behavior. For example, a Convoy behavior may request activation of a Leader assignment for a convoy leader which collects the trail traveled by the vehicle and transmits it to other vehicles. At the same time, it may also request activation of a Convoy assignment on all vehicles to facilitate inter-vehicle position updates to all members of a convoy.

Assignments register themselves with an Assignment Manager at startup, and behaviors request the activation of one or more assignments depending on the high-level mission goals. Behaviors may sequence the requests for activation/de-activation of assignments to perform complex or multi-stage behaviors.

The Assignment Arbiter monitors the assignment selection state, and when multiple assignments are active, arbitrates between them to determine which assignment outputs to forward to the Planner Management System. Assignments request the activation of a planner in the navigation management system and provide the planner with an objective point or path to achieve.

Individual assignments may also send and receive messages to other subsystems, such as the world model or inter-vehicle communications.

The Assignment Management System has the same structure as the Behavior Management System: An Assignment Manager and Assignment Arbiter manage the activation and output of one or more assignment modules.

The Assignment Manager subscribes to activation requests from the Behavior Manager and uses those requests to enable and disable the execution of assignments. Unlike the Behavior Management system, one or more assignments may execute in parallel. This gives rise to complex behaviors, as Assignments can share and coordinate information with each other, as well as with other elements of the system. Assignments that control vehicle motion publish at least two messages: a path or goal objective for a planner, and an activation request for a planner to act on that objective. Assignments that perform tasks that do not affect vehicle movement (e.g. inter-vehicle communication, or recording paths for followers or retro-traversal,) do not have to publish either of those messages.

The Assignment Arbiter accepts the output of individual assignment modules and re-publishes the path or goal objective of the assignment with the highest execution priority to the Planner Manager component.

Planner Management System

Planner Components, or planners, provide increasing resolution to the goals published by Assignments. Planners must determine a navigable path to a commanded destination, avoiding both moving and stationary obstacles as reported by the World Model.

An implementation may contain multiple planners, each with its own unique application. For example, some planners may operate better in open-world conditions while others specialize in road following. Since multiple planners may be present, a single Planner Manager is responsible for selecting which planner is active at any given time. This is like the Behavior Management mechanism used to select and activate its different Behavior Modules.

In addition to the target trajectory, an active planner must also generate a "safe harbor" plan. For small slow vehicles, this may simply be a command to immediately stop using zero velocity. For larger, more complex platforms, the safe harbor is a trajectory that will bring the vehicle to a controlled stop and quickly and safely as possible such as using the first points of the commanded path to decelerate or continuing along the current heading until stopped. The system is never expected to execute the safe harbor plan, but it may be needed if failures or fault conditions arise.

The Planner Management System has the same structure as the Behavior Management System: A Planner Manager and a Planner Arbiter manage the activation and output of one or more Planner modules.

The Planner Manager subscribes to activation requests from the Assignment Manager and uses those requests to enable and disable the execution of Planner modules. Only one Planner module may be active at a time. Planner modules all accept a goal point or path, but each planner must define the semantics of that input (e.g. does the planner plan to a point or a path, should the desired path lie along the vehicle's current position). It is expected that assignments select a planner with the needed capabilities to execute the desired motion and provide semantically correct inputs as defined by that planner.

Planners are responsible for requesting the activation of an appropriate motion executor to execute the desired path. Each planner should use the available world model information from the World Model Component (e.g. costmaps) to plan a navigable path. Planners should each request an appropriate costmap from the World Model component, either during initialization or when activated.

The Planner Arbiter accepts motion executor activation requests, path commands, and safe harbor paths from all planner modules and re-publishes output from the currently activated planner to the Motion Execution system.

Motion Executor Management System

The Motion Executor (ME) Components are the lowest levels of control within the A-Kit, converting the navigable plans from the active Planner Component to actionable commands to the B-Kit. Motion Executor output is based on motion primitives, such as commanded velocity and curvature (or steering commands) that are easily executed by the B-Kit.

Like the Planner Components, multiple Motion Executors can exist in a single implementation. A single Motion Executor Manager selects and activates a specific ME based on the autonomy mode and the capabilities required. Note that this structure is largely for future proofing, as generally systems will have only a single ME to convert commanded plans to motion primitives based on the capabilities of the vehicle.

In addition to decomposing the active plan into motion primitives, the active Motion Executor must also forward the safe harbor plan. This plan is passed through unaltered; hence, the B-Kit itself must be capable of executing the safe harbor plan in the event it becomes necessary.

The Motion Executor Management System has the same structure as the Behavior Management System: A Motion Executor Manager and a Motion Executor Arbiter manage the activation and output of one or more Motion Executor modules.

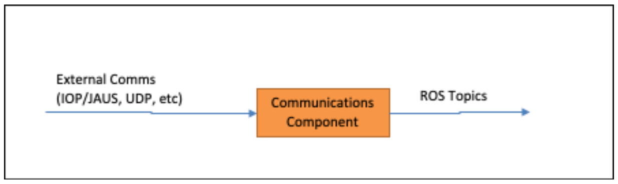
The Motion Executor Manager requests the activation of a single Motion Executor Module based on the request from the Planner Arbiter. Only one motion executor can be active at a time.

Each Motion Executor module is responsible for translating an input path into a set of primitive commands for the underlying platform to execute. The exact format of these motion primitives may vary depending on the capabilities of the underlying platform controller. Motion Executors are also responsible for forwarding a Safe Harbor command to the underlying platform.

The Motion Executor Arbiter accepts outputs from each Motion Executor and forwards the commands from the active Motion Executor to the underlying platform.

Communications Component

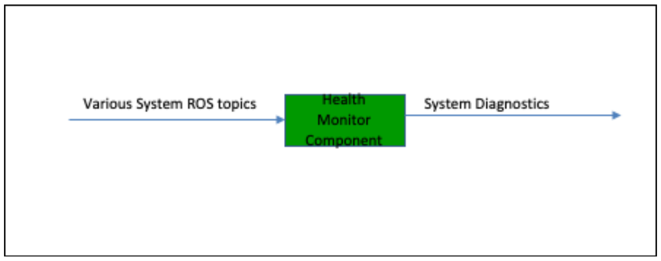
Data exchange, whether OCU-to-vehicle, vehicle-to-vehicle, or A-Kit to B-Kit, is expected to use IOP. Internal to the A-Kit, however, messaging is based on ROS2. The Comms Model Component, sometimes called an IOP-to-ROS bridge, converts between these two protocols. As such it becomes the gatekeeper for information flowing into and out of the A-Kit.



Since the Comms Module Component is a translator and does not generate or consume messages on its own, specific inputs and outputs are not enumerated. Rather, the breadth of the bridge is based on the message set of the A-Kit itself as well as the particular IOP Instantiation defined for the mission. As such, the input and output messages consist of any messages specified by the IOP Instantiation for the specific project or program realizing this architecture, and the ROS messages defined therein.

Health Monitor and Diagnostics Component

The Health Monitor and Diagnostics Component monitors the overall health of the system. Consequently, it can be configured to monitor the message traffic from virtually any component in the A-Kit and detect problems that prevent safe and normal operation. Any such problems are reported out to the OCU for display to the human operator, and also reported to the Behavior Manager to properly manage any necessary state transitions associated with the error.

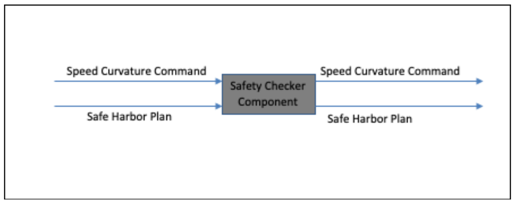


AGR in particular uses a sophisticated scripting language to configure the specific health checks and resulting behavior of the Component. While such an approach leans more towards implementation than architecture, it is noted that such an approach offers significant flexibility at run-time without the need to modify and recompile code. As a result, implementations are encouraged to consider this solution.

The capabilities of the Health Monitor are largely dependent on the specific program, vehicle, and common failure modes. Because of that, the design constraints specified by this architecture are minimal. The implementation may subscribe to any message within the architecture and monitor for warning and error conditions.

Safety Checker Component

The Safety Checker is an optional component for safety critical platforms and programs, generally involving larger and/or faster vehicles with significant potential for human injuries or fatalities in the event of a failure. One and only one Safety Checker Component is expected per vehicle.



The Safety Checker Component runs on dedicated hardware, and serves as the final gatekeeper between the Motion Executor Components and the B-Kit. In the event a failure or unsafe condition is detected, the Safety Checker can alter the commanded path or force execution of the safe harbor plan. It is expected that the Safety Checker implementation would be certified by relevant DoD authorities.

The Safety Checker is largely a pass-through component, providing a safety-critical validation of the Motion Executor commands between it is passed to the B-Kit for execution on the platform. Consequently, the inputs and outputs are the same message structure; however, the message data may be modified by the Safety Checker if an error condition is detected. Note that the publication rate is expected to match that of the incoming message rate. Further, the Safety Checker must not cause significant latency of the command stream.

6. CONCLUSION / PATH FORWARD

The project to develop an initial design for a common, open, Government-owned, autonomy software architecture has resulted in a promising prototype with the potential to provide the flexibility needed by the both the Government and industry RDT&E communities to leverage reusable software and more rapidly innovate new capabilities; while ensuring the discipline and enabling the modularity required to develop RDT&E software structured to meet the software safety, cybersecurity, upgradeability, and other needs of RAS programs of record. The next steps for further advancing the prototype SCION architecture include the following:

- ➢ Leveraging the AGVRA libraries and meta-models: capture the system requirements in SysML, complete a detailed design of the SCION architecture, and capture the detailed design in SysML
- ➢ Defining, capturing, and modeling the interfaces and data flow among the components in the SCION architecture, using the AGVRA Data Interoperability Architecture (DIA) artifacts and meta-models
- ➢ Developing a ROS2 interface definition set for the SCION architecture leveraging the newly created SysML models and inter-component data definitions
- ➢ Determining the vehicle and sensor characteristics of a target RAS platform and creating an AGVRA physical model viewpoint of the system
- ➢ Investigating and analyzing how elements of other autonomy software architectures might be incorporated into future versions of the SCION architecture and develop a long-term roadmap
- ➢ Investigating whether and how the RTK packages converted to ROS2 under the MARS project might be subsequently migrated to the SCION architecture