Version 1.0.0

Overview, Governance, Requirements, Design, & Implementation

# InterUSS Platform™

Open Source Project

## Objective

This document provides the philosophy, design, implementation, governance, and guidelines for the platform used by separate UAS Service Suppliers (USS) to communicate during UAS operations, known as the InterUSS Platform™. This flexible and distributed system is used to connect multiple USSs operating in the same general area to share safety information while protecting competition among USS's, confidential business information, and operator and consumer privacy. The system is focused on facilitating communication amongst actively operating USSs with no details about UAS operations stored or processed on the InterUSS Platform.

## Background

For the past few years, Project Wing has been a proponent for the development of federated, industry-led airspace management solutions to facilitate the safe operations of UAS in our skies.

As part of NASA's TCL3 testing in April 2018 at the Mid Atlantic Aviation Partnership test site in Virginia, flights flown using Project Wing's and AirMap's UTM platforms were safely routed around each other using an early version of the InterUSS Platform - demonstrating that UAS can safely share the skies regardless of the UTM provider the drone is using. The success of TCL3 is proof that strong cooperation between private and public industry players is essential to enabling the safe and scalable operation of drones in low-altitude airspace, and is already happening.

In order to continue this progress and move from local demonstration to global reality, Project Wing is open sourcing the InterUSS Platform design and code, making it available to a community of USSs that can diversify the capabilities and accelerate adoption in the real world.

# Table of Contents

## InterUSS Platform Philosophy

To best support collaboration among USSs, there are two key tenets to the InterUSS Platform.

1. The InterUSS Platform is focused on **facilitating communication** amongst **actively operating** USSs.

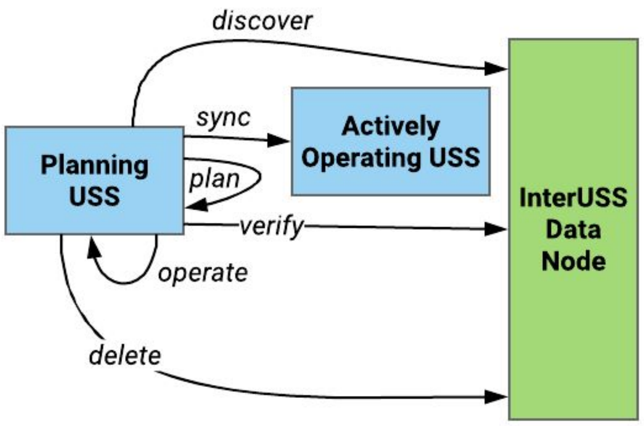
2. **No details about UAS operations** are stored or processed on the InterUSS Platform.

Regardless of the new features and changes brought to the platform via the open source process, these two tenets must be enforced to ensure protection of operator and consumer privacy and ensure the competition between USSs for drone services remains intact.

## Concept of Operations

When a USS wants to plan a flight, the “planning USS” performs the following steps:

1. **Discover** - Communicates with the InterUSS platform to discover what other USSs have an active operation in the specific area of flight and how to contact them. Flight details are not stored in the InterUSS Platform for both minimizing data bandwidth and maximizing privacy.
2. **Sync** - If there are other USSs with active operations, the planning USS contacts the other USS(s) with active operations in the area and retrieves the operational information necessary to de-conflict the operation.
3. **Plan & Verify** - Once the operation is safely planned and deconflicted, the USS updates the InterUSS platform with their contact information for other USSs to use for future discovery. This update also provides final verification that the planning was done with the latest operational picture.
4. **Delete** - Once the operation is completed, the USS removes their information as they do not need to be contacted in the future since the USS no longer has operations in that specific area of flight.

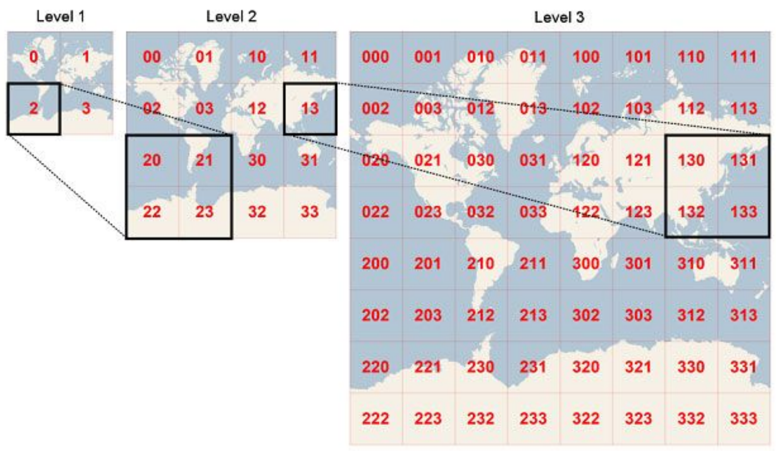


All of these operations are secured using a standard OAuth communications flow, authorized by a governmental authority or, in the absence of an authority in certain parts of the world, industry provided OAuth solutions decided by the InterUSS Technical Steering Committee.

## Technical Overview

The InterUSS platform is a **distributed consistency**, **pull-based, gridded** system used to share information:

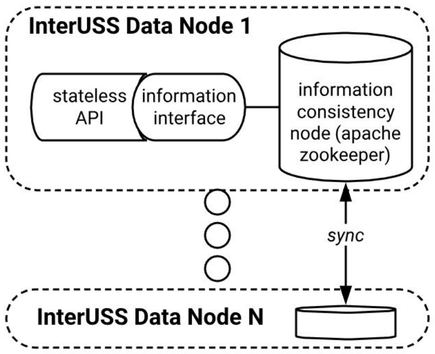
- **Grid system** - the world space is separated into a well known grid format, minimizing contention and over-sharing, while allowing flexibility as the number of USSs and countries with UTM systems grows.



Level 15 ~780m
Santo Domingo
Gran Vía
Callao
Calle de Bailén
Ópera
Calle Arenal
Sol
Madrid
Calle Mayor
Barrio de los Austrias

- **Distributed consistency** - multiple open source data consistency nodes result in a distributed, auditable, and flexible way to solve race conditions when multiple USSs are planning at the same time.

- **Pull-based** - flight information is acquired at the time of need, which protects operator and consumer privacy while sharing the right amount of information to safely deconflict and inform multiple USSs.



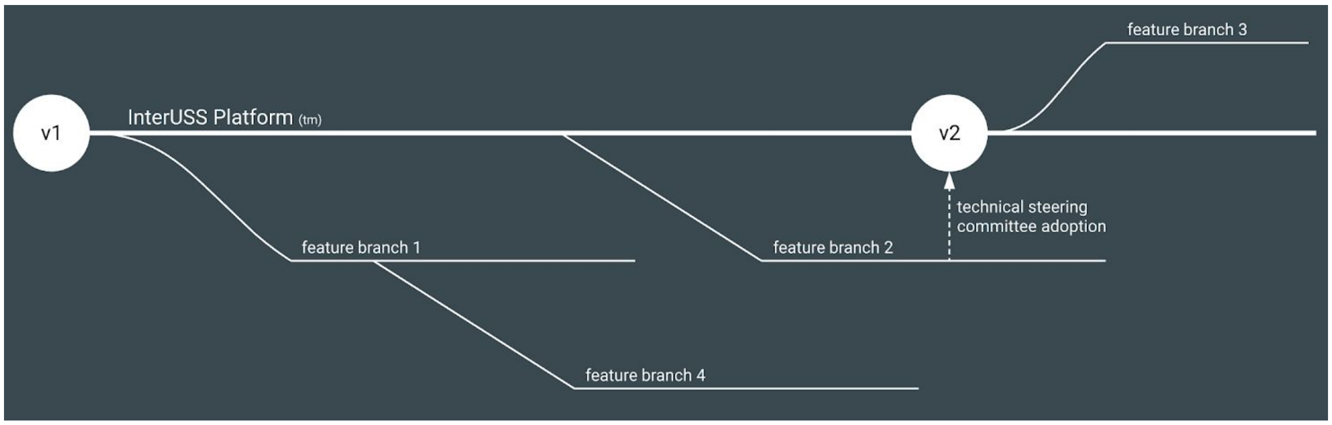
The open source implementation consists of the entire architecture of the platform: the **data consistency nodes** (Apache Zookeeper), the **authentication nodes** (Standard OAuth) and the **interface/API** (open source python) for interacting with the data about which USS is operating in which cells, and how to communicate with them.

**Have enough information and want to start playing?**
Head over to [https://app.swaggerhub.com/apis/InterUSS_Platform/data_node_api](https://app.swaggerhub.com/apis/InterUSS_Platform/data_node_api)
to try it live in a sandbox!

## Open Source Overview

The source code is completely open for contributions, modifications, and forking into new instances. This openness will allow other USSs and research agencies to branch the main baseline and modify design & code for particular use cases or demonstrations.

The trademarked InterUSS Platform is the main baseline code that follows a specific process for accepting patches & changes to the design & code (including rigorous testing), informed and guided by the InterUSS Platform steering committee, as illustrated below.



The InterUSS Platform open source project follows an open source technical charter based on the Linux Foundation for open source projects, drafted here. The community consists of **participants** (those USSs that use another USSs instantiation of the open source to communicate to other USSs), **hosts** (those USSs that use the open source and host one or more of the data consistency nodes required), **contributors** (those that can develop changes), and the **technical steering committee members** (those that have a vote on changes and may host an authentication node and/or data consistency nodes).

The source code, documentation, and docker image(s) will be hosted on GitHub and has an Apache 2.0 License. We'd love to accept your patches and contributions to this project. There are just a few small guidelines that need to be followed.

Contributions to this project must be accompanied by a Contributor License Agreement. You (or your employer) retain the copyright to your contribution, the CLA simply gives us permission to use and redistribute your contributions as part of the project. Head over to https://cla.developers.google.com/ to see your current agreements on file or to sign a new one. You generally only need to submit a CLA once, so if you've already submitted one (even if it was for a different project), you probably don't need to do it again.

All submissions to the InterUSS Platform baseline, including submissions by project members, require review. We use GitHub pull requests for this purpose. Consult GitHub Help for more information on using pull requests.

## Roadmap

Multiple activities will continue to branch and enhance the platform. In addition to the below initial release requirements, parallel activities currently underway include:

- **NASA's TCL4 Demonstration** - additional services can be supported by updating the metadata information to include a USS's base URL for USS Required API[1].
- **Public Portal** - additional API endpoint(s) to support sharing position information for public portals to show drones from multiple USSs.

Additionally, other use cases may support other features outside of the mainline branch, like different authentication methods or grid formats. Final timing of acceptance of these features into the InterUSS Platform will be decided by the results of the demonstrations and the Technical Steering Committee.

### Initial Release Requirements

The following capabilities are addressed in the first version of the source code released:

- **Provide the capability for an authorized and authenticated USS to obtain USS information about drone operations for the purposes of cooperative deconfliction.**
    - *This requirement specifically says "USS information about drone operations" which does not need to include the details of the operations themselves.*
    - *This requirement implies authorization and authentication processes are part of the project.*
- **Provide assurance information about drone operations is up-to-date and consistent, even when multiple operations are being deconflicted at the same time from multiple USSs.**
    - *This requirement should result in no race conditions or other data inconsistency operations could occur that could jeopardize safety.*
    - *This requirement implies there is a way to confirm when local information is out of date.*
- **Provide assurance the platform can protect operator & consumer data by minimizing information sharing to specific areas required for cooperative deconfliction.**
    - *This requirement should result in splitting up information to only be shared when there is an overlap*
- **Provide assurance the platform can scale as more drone flights and more USSs join the open source project.**
    - *This implies horizontal scaling, i.e. more nodes not more horsepower.*
- **Provide the definitions and the reporting capabilities on SLAs, including performance, usage, and abuse.**

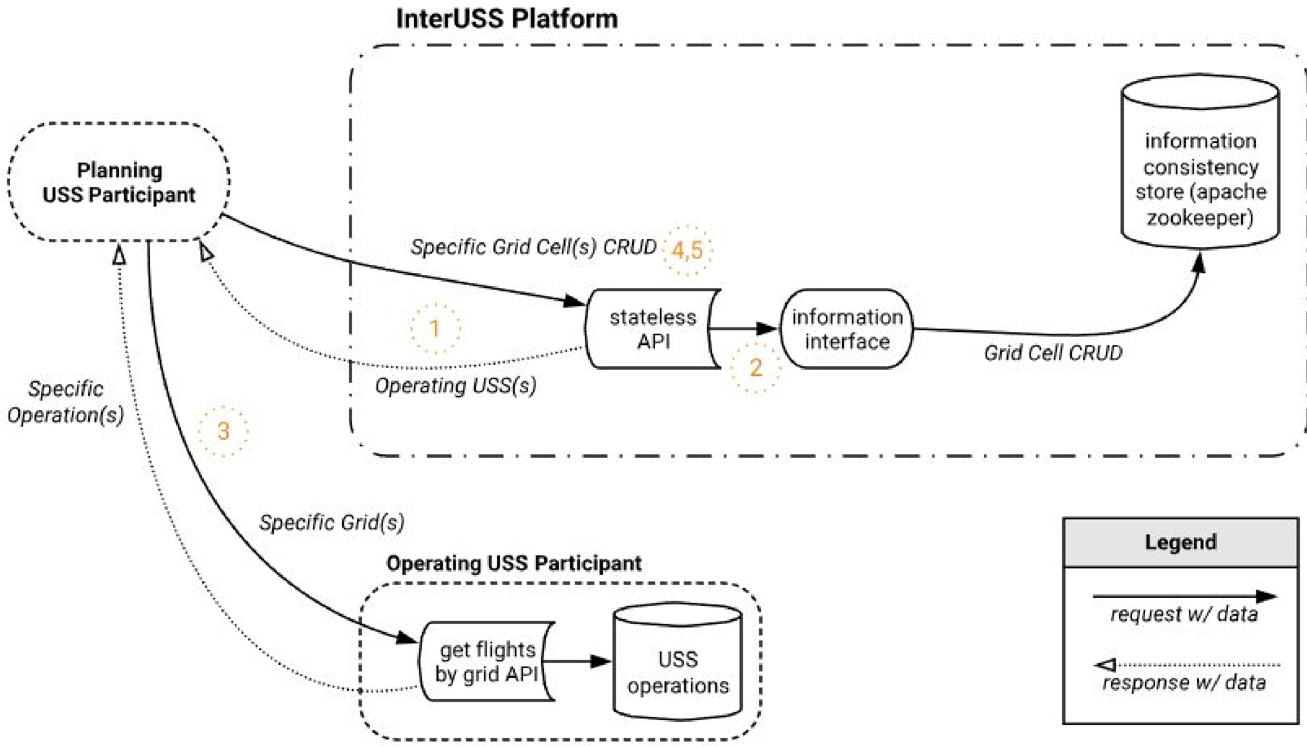
[1] https://app.swaggerhub.com/apis/utm/uss

## Architecture

The following diagrams provide various views, interactions, and implementation of the overall architecture of the InterUSS Platform. Together they represent the InterUSS Platform architecture which facilitates communication amongst actively operating USSs.

### Logical Architecture

The first view of the architecture is the logical view, showing the invocation process and data flow between the participants and components of the platform, illustrated below. The authorization and authentication process is purposefully omitted from this view for simplicity, and shown in a later view.



The abbreviated Logical Architecture of the InterUSS Platform (hiding the authentication / authorization flow) describes the operations a USS would perform to plan a new flight with cooperative separation.

Expanding the illustration, the invocation and data flow is as follows:

1. Invocation starts with the Planning USS contacting the InterUSS Platform during the pre-planning or active re-planning phase of a UAS operation. The Planning USS provides the specific grid cell or cells the planned operation intends to intersect with and receives metadata information about the Operating USS(s) in those grid cells (if any) and metadata about the grid cell(s), such as the last time it was updated and a sync token to be used when updating the InterUSS Platform.

2. Within the InterUSS Platform, the information interface contacts the information consistency store for the particular grid cell(s), translating the request into the

proper format and methods for the underlying data infrastructure (currently Apache Zookeeper). Once the data is received from the information consistency store, the information interface formats the response in easy to digest JSON format and sends it back to the Planning USS.

3. Once the response is received, the planning USS contacts the operating USS(s) for information about the operations in the particular grid cell(s) where other USS(s) are operating. Those USS(s) respond with the details of the active or planned operations to allow the Planning USS to deconflict their operation.

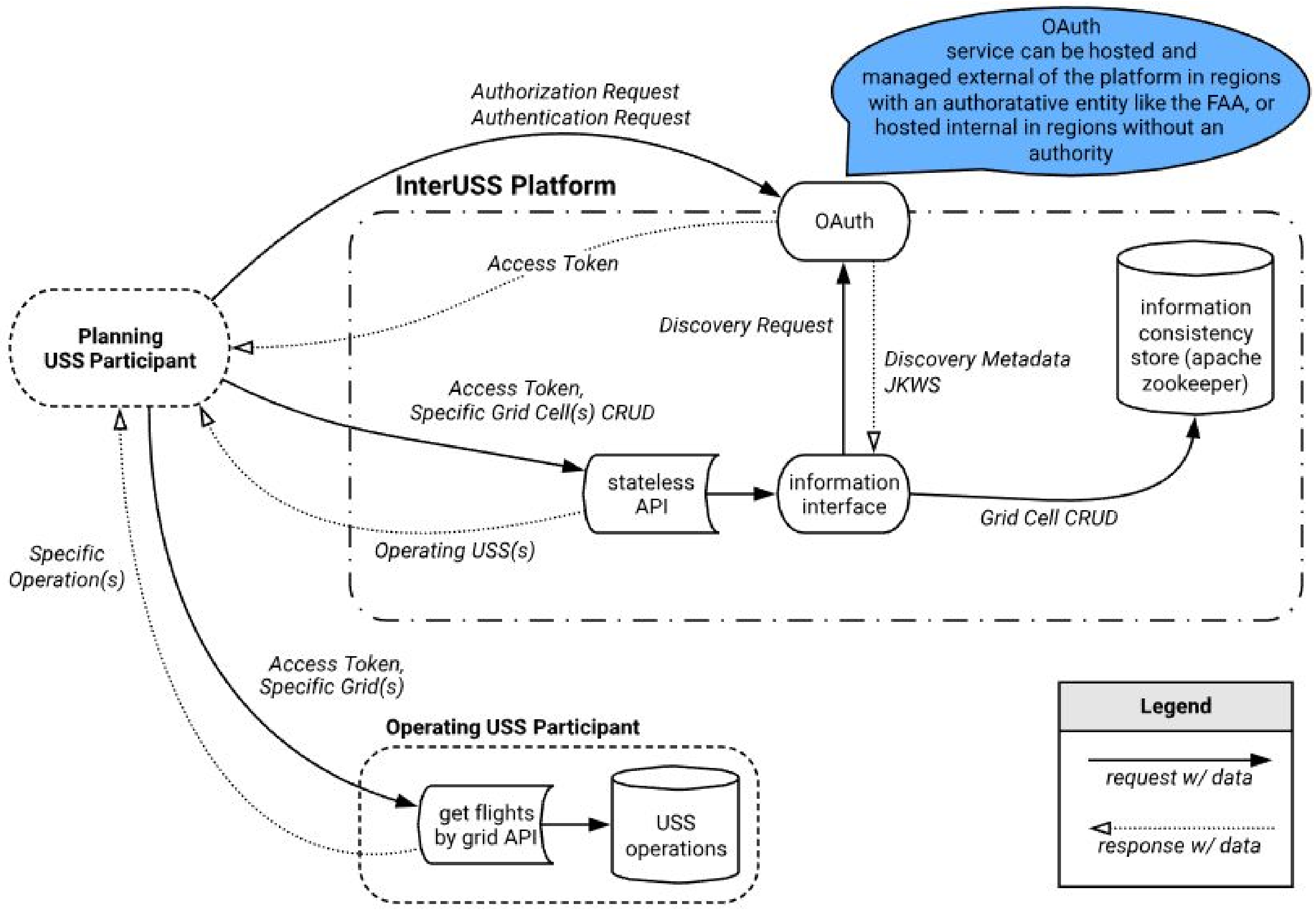
4. Once the Planning USS deconflicts their operation with all known other operations, they contact the InterUSS Platform with a write operation, providing their metadata information for grid cell(s) they will be operating in and the sync token they received during the read operation. If another USS has updated grid cell(s) during the deconfliction process, the InterUSS Platform will reject the update and allow the Planning USS to repeat the deconfliction process. Once the write operation is successfully completed, the Planning USS may proceed with their UAS operation and serve the details of the operation when asked by other USSs.

5. Once all of the operations are completed in a grid cell, the USS will contact the InterUSS Platform with a delete operation, which will remove their entry for that grid cell. This delete operation updates the metadata about the grid cell but does not update any other USS entries.

During this process, note that no UAS operation data is stored or processed on the InterUSS Platform. The platform is only focused on facilitating communication amongst actively operating USSs.

## Logical Architecture with Authorization/Authentication

The second view of the architecture is another logical view, including the authorization and authentication processes, illustrated below. The InterUSS Platform uses standard, off-the-shelf OAuth processes and software, with unique access tokens for interacting with the InterUSS Platform and each Operating USS.


OAuth service can be hosted and managed external of the platform in regions with an authoratative entity like the FAA, or hosted internal in regions without an authority
Authorization Request
Authentication Request
InterUSS Platform
OAuth
Access Token
Planning USS Participant
Discovery Request
Discovery Metadata JKWS
information consistency store (apache zookeeper)
Access Token, Specific Grid Cell(s) CRUD
stateless API
information interface
Grid Cell CRUD
Operating USS(s)
Specific Operation(s)
Access Token, Specific Grid(s)
Operating USS Participant
get flights by grid API
USS operations
Legend
request w/ data
response w/ data

The full Logical Architecture of the InterUSS Platform describes the operations a USS would perform to obtain the necessary authorization/authentication and plan a new flight with cooperative separation.

In the context of this diagram, “users” are USS Participants, not end-users. Expanding on the logical architecture from the previous view, this view includes the authorization process (approving/adding/rejecting/removing of users of the system) and authentication process (validating users and providing access to the InterUSS Platform and other USS systems).

Once a user has passed the process for becoming an authorized InterUSS Platform Participant *(process TBD based on country and technical steering committee)*, the participant may request authorization to the OAuth node for areas they wish to participate in. Once approved, the authentication credentials are established, synchronized amongst all OAuth systems, and provided to the participant.

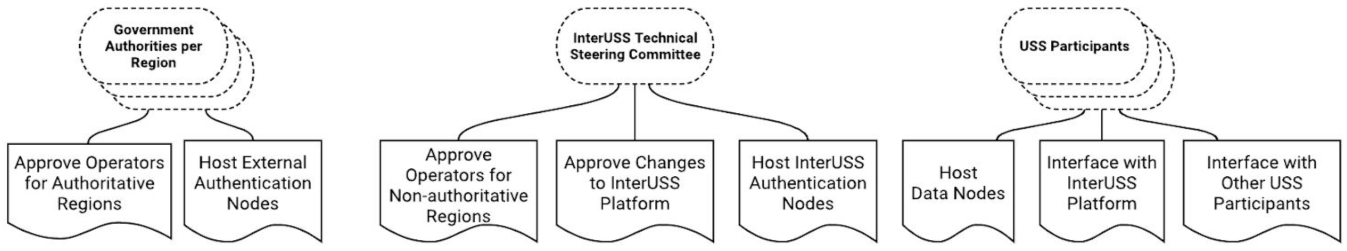
After receiving the credentials, the participant can now use the credentials to obtain an access token to the InterUSS Platform scope, valid for a predetermined amount of time. This token is used during the requests described in the previous section. The InterUSS Platform uses OAuth discovery mechanisms to obtain public keys from valid OAuth hosts. These keys are used for decrypting access tokens without requiring an active interface to the OAuth host. Combined with the validity time of the token, this enhances resilience to OAuth outages.

Once another operating USS is identified in a grid cell, the Planning USS must request an authorization token to contact that particular USS. The process uses the same credentials as previous, but obtains the authentication token using the Operating USS scope. This access token has its own validity time and can be reused until it expires.

## Physical Architecture

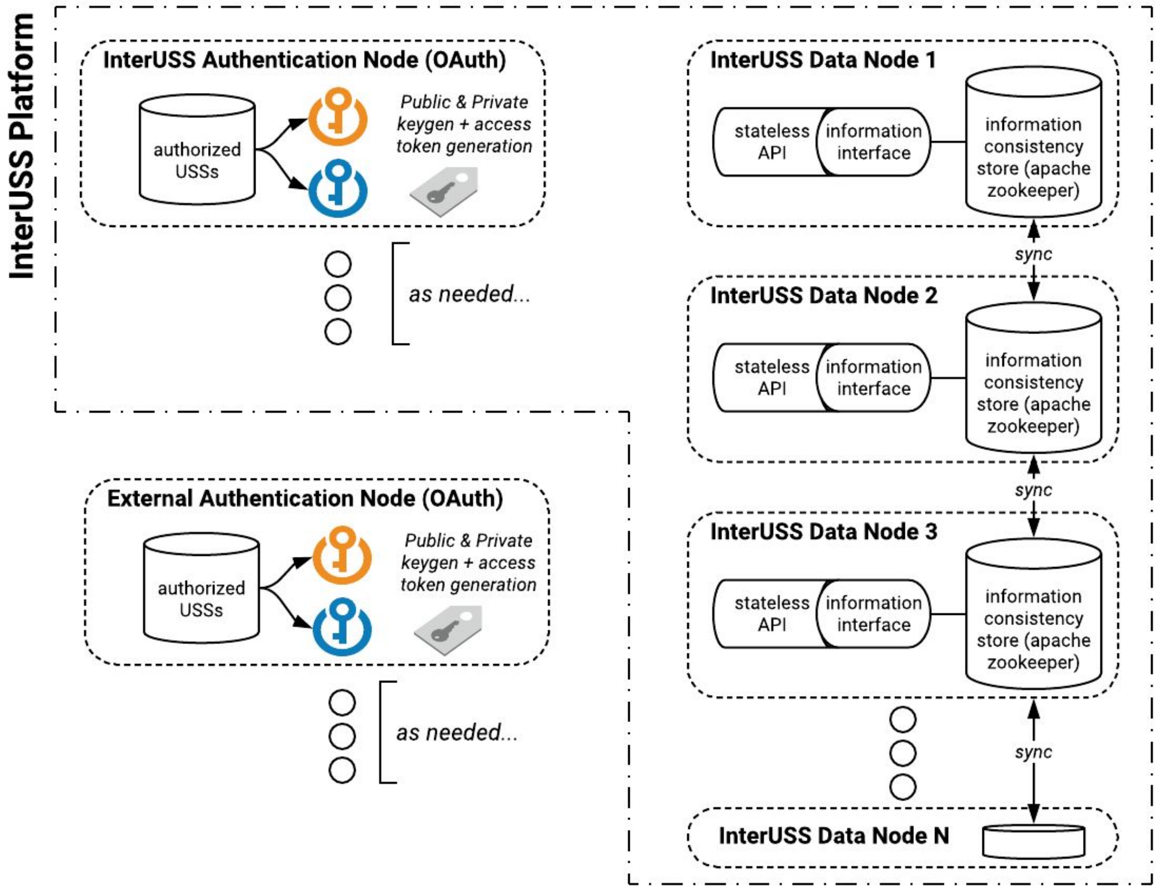
The next view of the architecture is the physical view, illustrating the roles for the platform, the encapsulation the logic into deployable systems, and mechanisms for scaling the InterUSS Platform.

The roles for the platform are filled by three entities, the USS Participants, the InterUSS Platform Technical Steering Committee, and the Government Authorities (as available per region).  Their responsibilities are illustrated below.


Government Authorities per Region
Approve Operators for Authoritative Regions
Host External Authentication Nodes
InterUSS Technical Steering Committee
Approve Operators for Non-authoritative Regions
Approve Changes to InterUSS Platform
Host InterUSS Authentication Nodes
USS Participants
Host Data Nodes
Interface with InterUSS Platform
Interface with Other USS Participants

Depending on the region, a government authority may approve USSs for access to the InterUSS Platform and may host External Authentication Nodes, outside the InterUSS Platform. In a region without an active government authority, the InterUSS Platform Technical Steering Committee may approve USSs for access to the InterUSS Platform and may host Authentication Nodes inside the InterUSS Platform. Finally, USS Participants may host Data Nodes, and will interface with the InterUSS Platform and other USS Participants.

The InterUSS Platform consists of the **InterUSS Data Node**, which contains the API, logic, and underlying data structures that hold and access the information about how and when to communicate with other USS Participants and the **InterUSS Authentication Node**, the mechanism to obtain credentials for USS Participants to securely communicate to the InterUSS Data Node and other USS Participants, which can be hosted internal to the platform by USSs, or externally hosted by a government authority.


InterUSS Platform
InterUSS Authentication Node (OAuth)
authorized USSs
Public & Private keygen + access token generation
as needed...
External Authentication Node (OAuth)
authorized USSs
Public & Private keygen + access token generation
as needed...
InterUSS Data Node 1
stateless API
information interface
information consistency store (apache zookeeper)
sync
InterUSS Data Node 2
stateless API
information interface
information consistency store (apache zookeeper)
sync
InterUSS Data Node 3
stateless API
information interface
information consistency store (apache zookeeper)
sync
InterUSS Data Node N

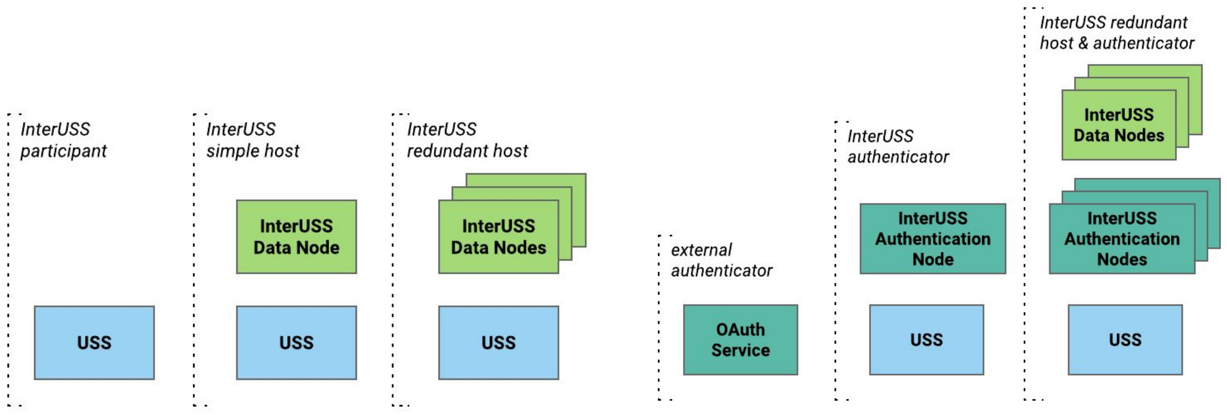
The Physical Architecture describes the responsibilities of the various parties and the physical implementation of the various components of the InterUSS Platform.

The **InterUSS Data Nodes** are encapsulated in a docker image for easy deployment and scaling. Within the nodes, the **information interface** communicates locally to the **information consistency store** (currently Apache Zookeeper). This allows tighter security for the Zookeeper instance, allowing read and write only from within the same data node. The Zookeeper instances synchronize with other Zookeeper instances. The allowable Zookeeper instances are configuration controlled by the Technical Steering Committee, and the configuration is distributed across all nodes whenever a new USS is allowed to host, or when a USS increases the number of nodes.

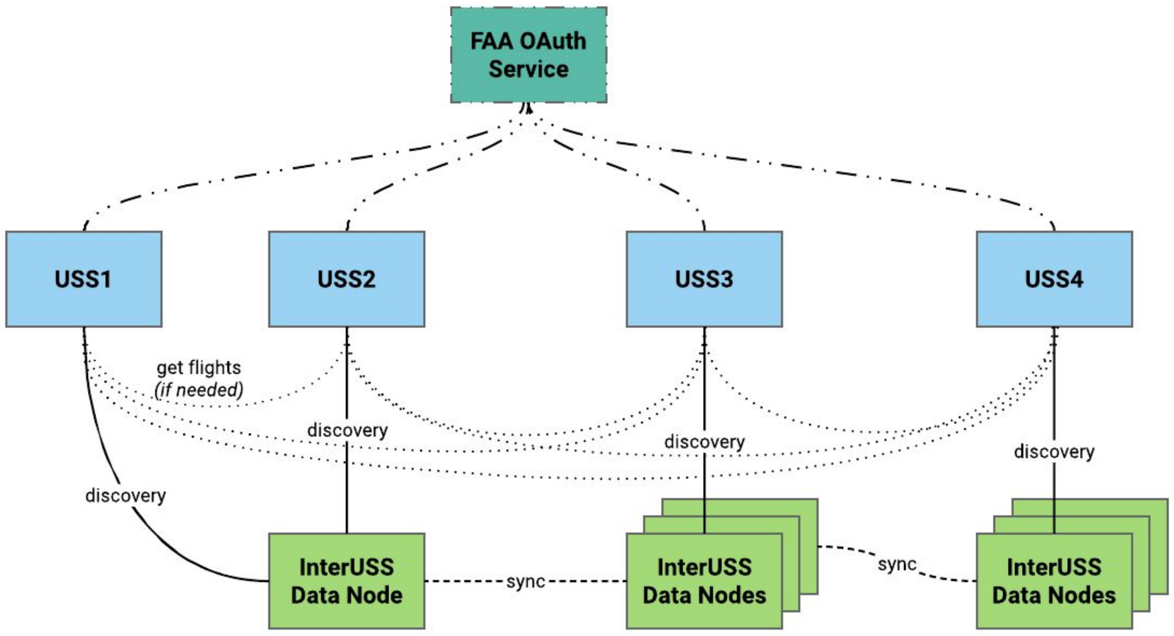
The **stateless API** provides a RESTful interface to creating, reading, updating and deleting information within a grid cell. The **InterUSS Authentication Nodes** are standard OAuth services, internally hosted by USSs, or externally hosted by a government authority, depending on region.

## Implementation Architecture

Based on the physical architecture, the actual implementation results in the roles illustrated below. This flexibility allows USSs and government authorities various levels of commitment to the InterUSS Platform community.



For example, in the US, an implementation may look like the illustration below, where the FAA is hosting the OAuth service, USS1 is only a participant and uses USS2's data node, USS2 is a simple host with one data node, and USS3 and USS4 are redundant hosts with multiple data nodes each.



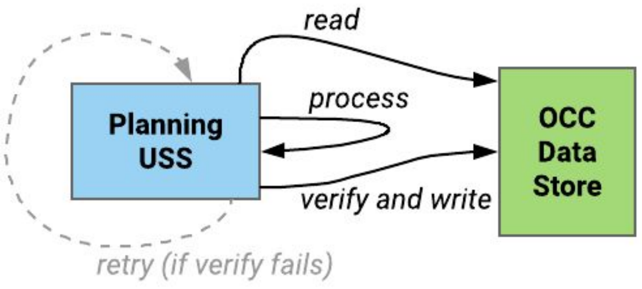
## Design Approach and Decisions

This section describes the design approach and decisions made for the InterUSS Platform.

### Optimistic Concurrency Control

With multiple USSs providing deconfliction services globally, great care must be taken to ensure the data is up to date and consistent amongst parties. To achieve global data consistency while meeting performance levels, the world space is split into manageable grids (see next section) and an optimistic concurrency control approach[2] is used.

Optimistic concurrency control was selected over pessimistic (or blocking) control since the occurrence of multiple USSs planning operations in the same area is relatively rare compared to the total number of operations. This means transactions can complete without the expense of managing locks and without having transactions wait for other transactions' locks to clear, leading to higher throughput than other concurrency control methods. This is technically accomplished by using built-in methods to Apache Zookeeper to read, write, and synchronize data and forcing one or more of the USSs to retry in the event of concurrent operations.

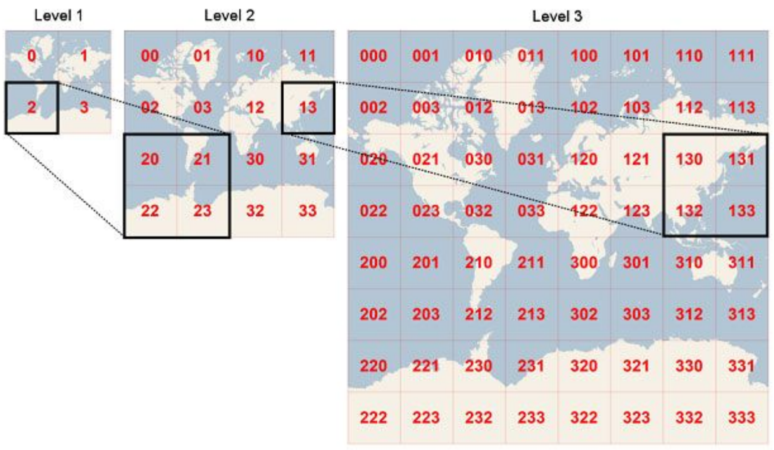


In the future, if Apache Zookeeper can no longer meet the needs of the InterUSS Platform, the data store can be changed without redesigning the platform. The data store is wrapped in an information interface, which limits the amount of change required in the code.

### Grid Cells

To minimize data transmission amongst USSs and protect operator and consumer information, the world is split into a well known grid format called Slippy Tile Format[3]. Any format to split the world could be used, but Slippy was chosen for its openness, its simplicity in translation from latitude and longitude and its scalability for various sizes of areas. A simplified model of Slippy is illustrated and explained on the following page.

[2] https://en.wikipedia.org/wiki/Optimistic_concurrency_control
[3] https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames



Slippy works at the initial zoom level (level 1) by using Mercator projection onto a square, and splitting the square into four quadrants. To get to a more detailed level, the quadrants are split into four more tiles each (level 2). This continues until the desired level of fidelity is achieved.  For the purposes of strategic deconfliction, zoom level 15 will be used. This provides bounds of approximately 780m per side, minimizing contention and over-sharing. Multiple cells can be accessed for a single operation, depending on the bounds of the operation.

## Off the Shelf Software Components

In alignment with the openness required for multiple USSs across the globe to communicate, the default architectural choice for software is off the shelf rather than developed, and open source where possible. Those choices of software are described in this section.

### Apache Zookeeper

Distributed data across multiple applications and providers is not a new concept in software engineering. Each time a solution is implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Multiple large businesses, including Box, Rackspace, Yahoo![4],  have solved this need with the open source Apache Zookeeper solution. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Further information about how Zookeeper is used in the platform is described in the Component Design - Information Consistency section further in this document.

[4] https://cwiki.apache.org/confluence/display/ZOOKEEPER/PoweredBy

### GitHub

(requirement of google open source)

### Docker

(easy to deploy, easy to horizontally scale)

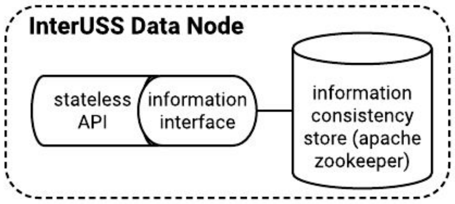
## Component Design

The design of the InterUSS Platform consists of three components:

- **InterUSS Data Node** - The API, logic, and underlying data structures that hold and access the information about how and when to communicate with other USS Participants.
- **USS Participant** - A USS that needs to share information with other USSs, both for planning a new operation and sharing existing operations when others are planning. Technically outside the InterUSS Platform, the participant interfaces with the platform and other USSs participating in the platform.
- **InterUSS Authentication Node** - The mechanism to obtain credentials for USS Participants to securely communicate to the InterUSS Data Node and other USS Participants. This is a standard OAuth service and will be authorized by a governmental authority or, in the absence of an authority in certain parts of the world, the InterUSS Technical Committee.

These three components are broken into sub-components explained in the following sections.

### InterUSS Data Node



A data node contains all of the API, logic, and data consistency infrastructure required to perform CRUD (Create, Read, Update, Delete) operations on specific grid cells. Multiple data nodes can be executed to increase resilience and availability. This is achieved by a stateless API to service USSs, an information interface to translate grid cell USS information into the correct data storage format, and an information consistency store to ensure data is up to date.

#### Stateless API

The API wrapper for reading and updating Grid Cells contains the following RESTful services:

- **GET GridCellMetadata** - provides an instantaneous snapshot of the metadata stored in a specific GridCell, along with a sync token to be used when updating. Returns 200 with a sync token and JSON metadata, or the nominal 4xx error codes as necessary.

- **PUT or POST GridCellMetadata** - updates the metadata stored in a specific GridCell using optimistic locking behavior, which ensures the data is consistent in the specific Grid Cell between reading and writing. The operation fails if unable to ensure data consistency or if the sync token has been updated since GET GridCellMetadata was originally called (based on sync token). Returns 200 and a new sync token if updated successfully, 409 if there is a data consistency conflict that could not be resolved, or the other nominal 4xx error codes as necessary.
- **DELETE GridCellMetadata** - removes the USS entry in the metadata stored in a specific GridCell using optimistic locking behavior, which acquires and releases the lock for the specific GridCell. The operation fails if unable to acquire the lock after a specific number of retries. Returns 200 and a new sync token if updated successfully, 409 if there is a data consistency conflict that could not be resolved automatically, or the other nominal 4xx error codes as necessary.

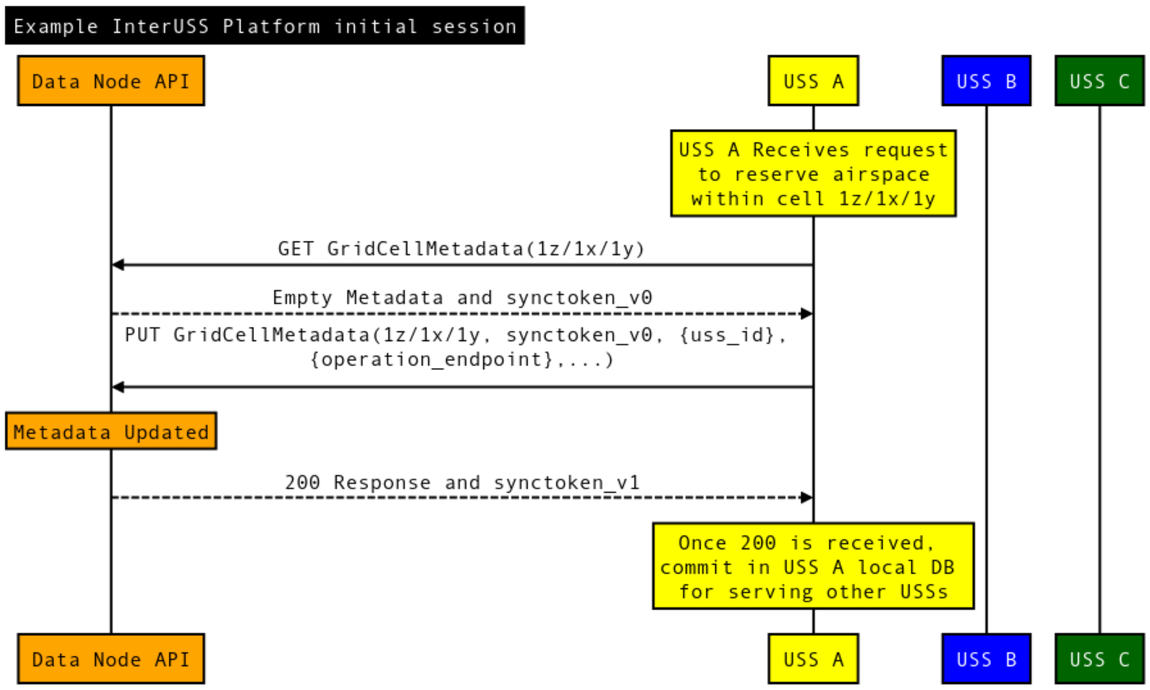
The format for the request and response is described in the SwaggerHub specification[5]. All responses follow the JSend format[6] for JSON information.

[5] https://app.swaggerhub.com/apis/InterUSS_Platform/data_node_api
[6] https://labs.omniti.com/labs/jsend

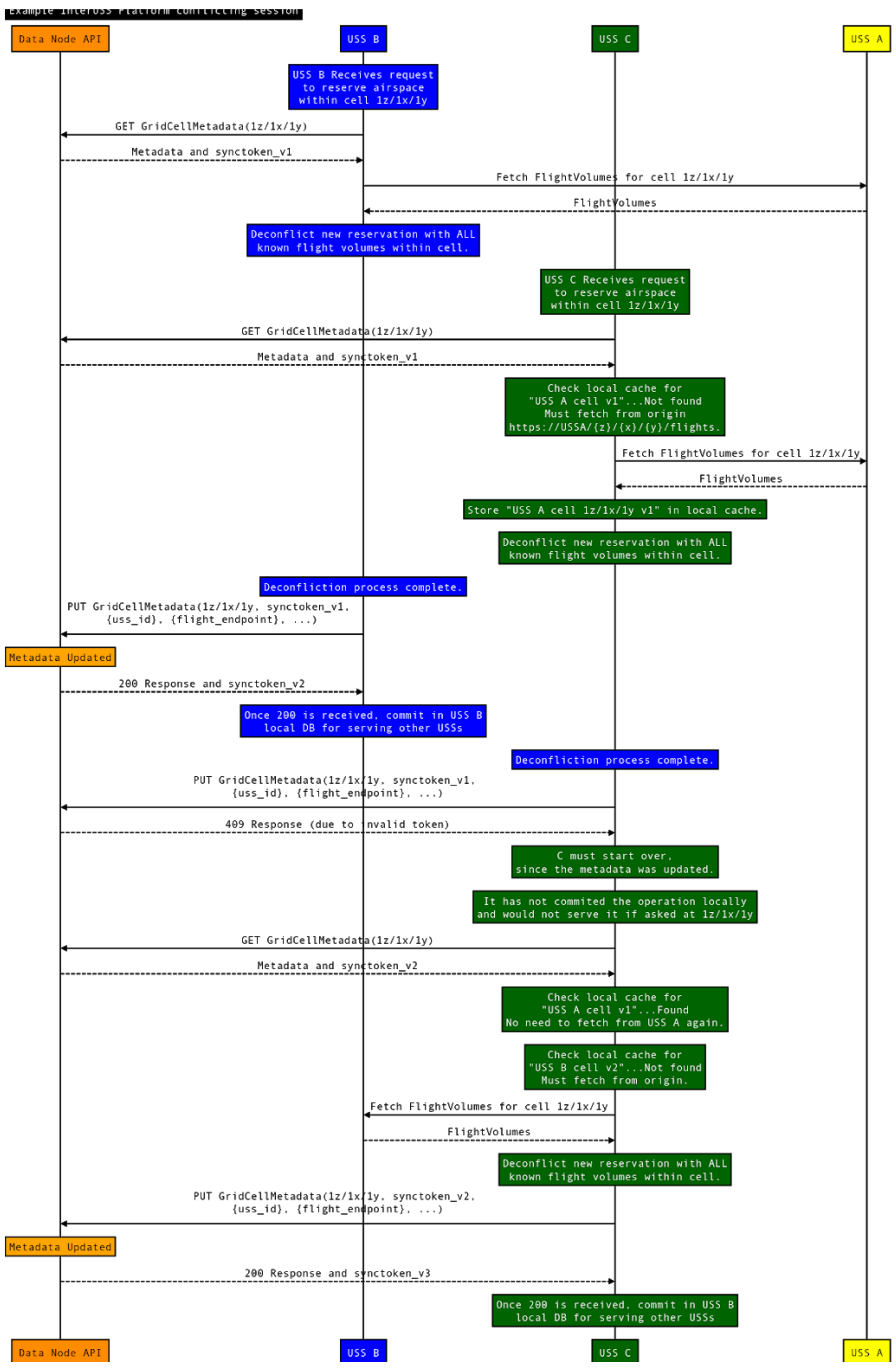
## Sequence Diagrams

Example sequence diagrams are illustrated in this section for three use cases:

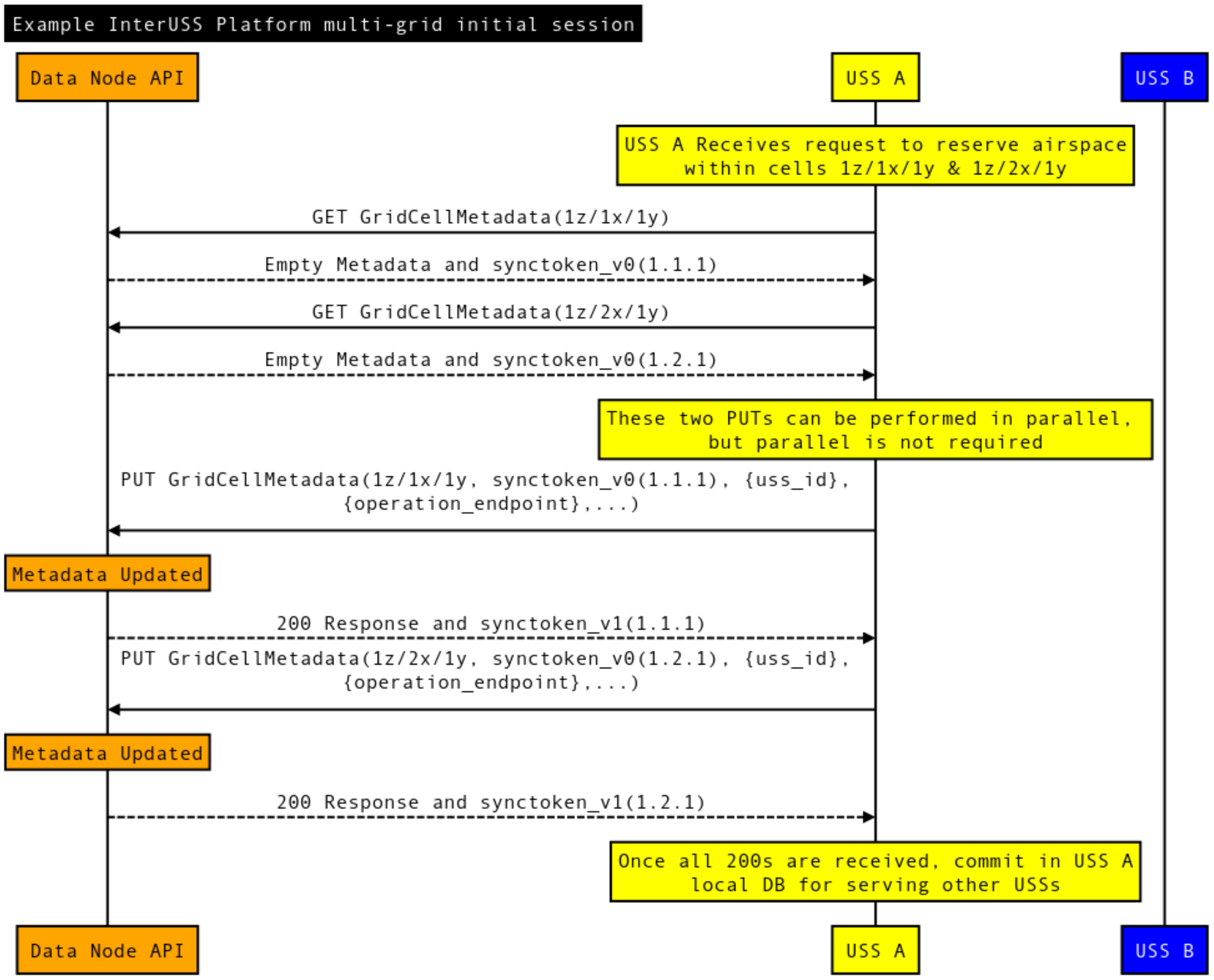
1. A successful planning operation with no other USSs in the grid cell,
2. A more complicated example of multiple USSs updating the same grid cell at the same time,
3. An example of a USS updating multiple grid cells for a single operation,
4. An example of a USS updating multiple grid cells and another USS updates one of the cells during the update,
5. Examples of the information concurrency protection built-in to the design.


Example InterUSS Platform initial session
Data Node API
USS A
USS B
USS C
USS A Receives request to reserve airspace within cell 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
Empty Metadata and synctoken_v0
PUT GridCellMetadata(1z/1x/1y, synctoken_v0, {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v1
Once 200 is received, commit in USS A local DB for serving other USSs
Data Node API
USS A
USS B
USS C

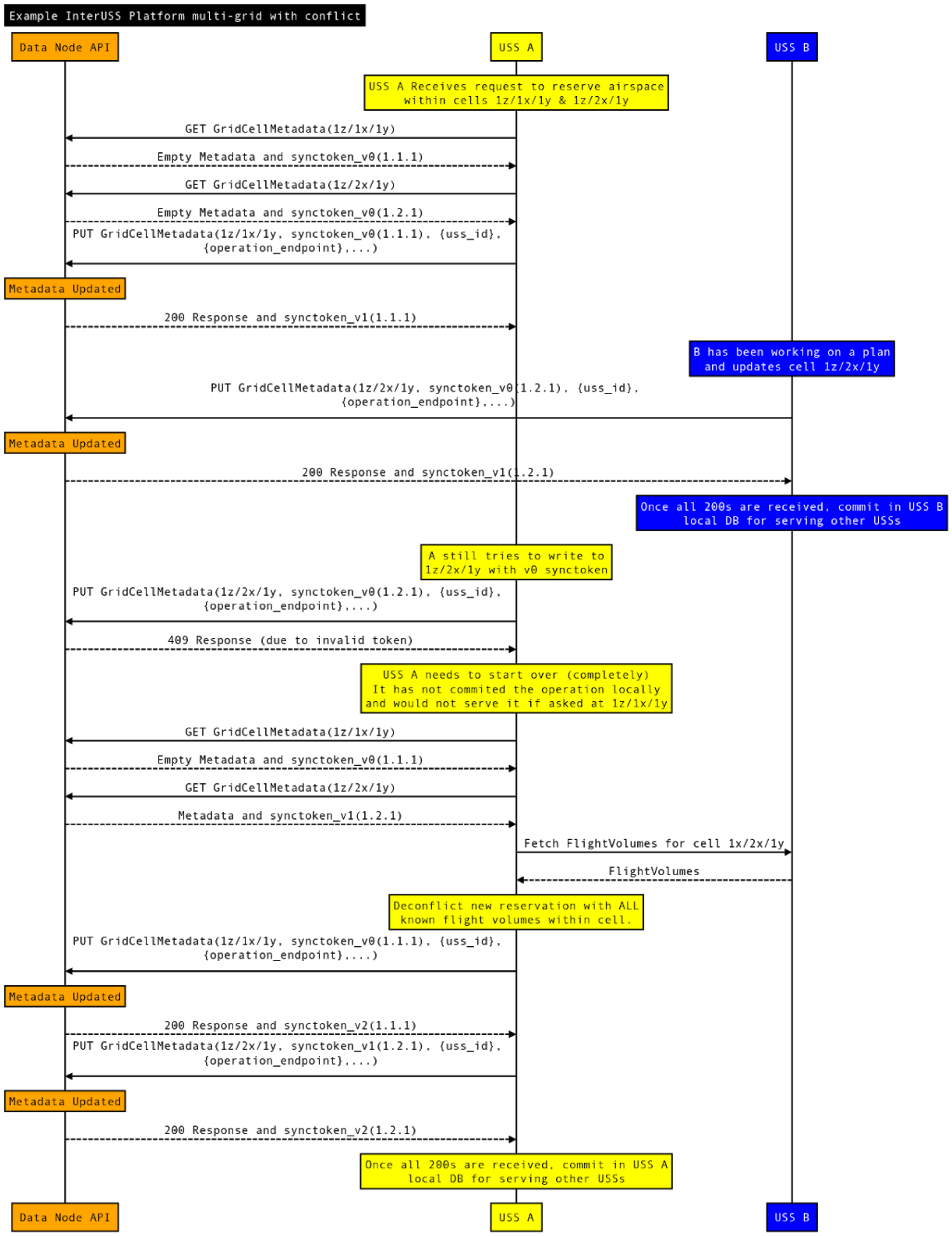
**Use Case 1:** a single USS planning an operation in a single grid cell.


Example InterUSS Platform conflicting session
Data Node API
USS B
USS C
USS A
USS B Receives request to reserve airspace within cell 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
Metadata and synctoken_v1
Fetch FlightVolumes for cell 1z/1x/1y
FlightVolumes
Deconflict new reservation with ALL known flight volumes within cell.
USS C Receives request to reserve airspace within cell 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
Metadata and synctoken_v1
Check local cache for "USS A cell v1"...Not found Must fetch from origin https://USSA/{z}/{x}/{y}/flights.
Fetch FlightVolumes for cell 1z/1x/1y
FlightVolumes
Store "USS A cell 1z/1x/1y v1" in local cache.
Deconflict new reservation with ALL known flight volumes within cell.
Deconfliction process complete.
PUT GridCellMetadata(1z/1x/1y, synctoken_v1, {uss_id}, {flight_endpoint}, ...)
Metadata Updated
200 Response and synctoken_v2
Once 200 is received, commit in USS B local DB for serving other USSs
Deconfliction process complete.
PUT GridCellMetadata(1z/1x/1y, synctoken_v1, {uss_id}, {flight_endpoint}, ...)
409 Response (due to invalid token)
C must start over, since the metadata was updated.
It has not commited the operation locally and would not serve it if asked at 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
Metadata and synctoken_v2
Check local cache for "USS A cell v1"...Found No need to fetch from USS A again.
Check local cache for "USS B cell v2"...Not found Must fetch from origin.
Fetch FlightVolumes for cell 1z/1x/1y
FlightVolumes
Deconflict new reservation with ALL known flight volumes within cell.
PUT GridCellMetadata(1z/1x/1y, synctoken_v2, {uss_id}, {flight_endpoint}, ...)
Metadata Updated
200 Response and synctoken_v3
Once 200 is received, commit in USS B local DB for serving other USSs
Data Node API
USS B
USS C
USS A

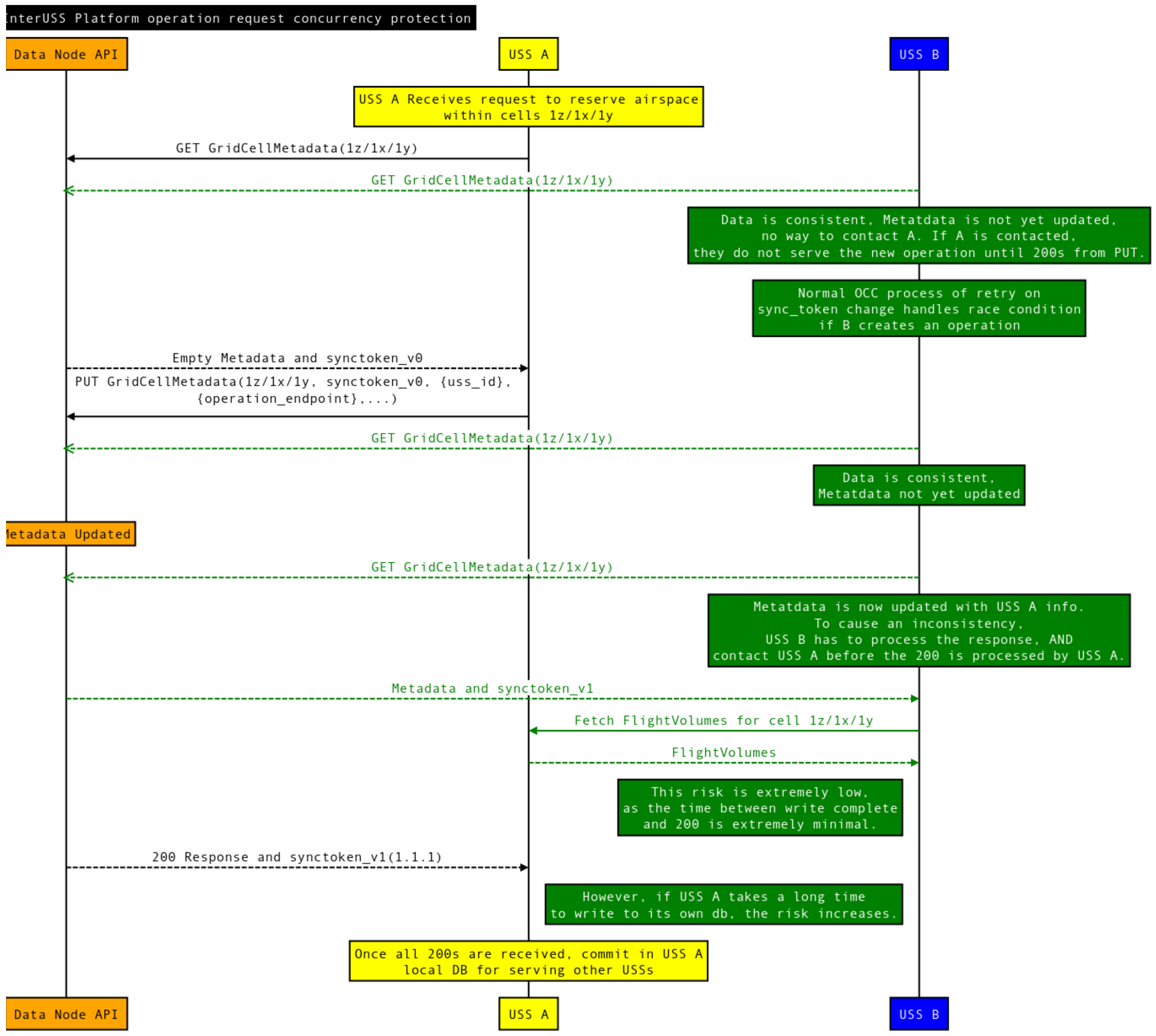
**Use Case 2:** multiple USSs planning operations in an already populated grid cell.


Example InterUSS Platform multi-grid initial session
Data Node API
USS A
USS B
USS A Receives request to reserve airspace within cells 1z/1x/1y & 1z/2x/1y
GET GridCellMetadata(1z/1x/1y)
Empty Metadata and synctoken_v0(1.1.1)
GET GridCellMetadata(1z/2x/1y)
Empty Metadata and synctoken_v0(1.2.1)
These two PUTs can be performed in parallel, but parallel is not required
PUT GridCellMetadata(1z/1x/1y, synctoken_v0(1.1.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v1(1.1.1)
PUT GridCellMetadata(1z/2x/1y, synctoken_v0(1.2.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v1(1.2.1)
Once all 200s are received, commit in USS A local DB for serving other USSs
Data Node API
USS A
USS B

**Use Case 3:** a USSs planning an operation in multiple grid cells.


Example InterUSS Platform multi-grid with conflict
Data Node API
USS A
USS B
USS A Receives request to reserve airspace within cells 1z/1x/1y & 1z/2x/1y
GET GridCellMetadata(1z/1x/1y)
Empty Metadata and synctoken_v0(1.1.1)
GET GridCellMetadata(1z/2x/1y)
Empty Metadata and synctoken_v0(1.2.1)
PUT GridCellMetadata(1z/1x/1y, synctoken_v0(1.1.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v1(1.1.1)
B has been working on a plan and updates cell 1z/2x/1y
PUT GridCellMetadata(1z/2x/1y, synctoken_v0(1.2.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v1(1.2.1)
Once all 200s are received, commit in USS B local DB for serving other USSs
A still tries to write to 1z/2x/1y with v0 synctoken
PUT GridCellMetadata(1z/2x/1y, synctoken_v0(1.2.1), {uss_id}, {operation_endpoint},...)
409 Response (due to invalid token)
USS A needs to start over (completely) It has not commited the operation locally and would not serve it if asked at 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
Empty Metadata and synctoken_v0(1.1.1)
GET GridCellMetadata(1z/2x/1y)
Metadata and synctoken_v1(1.2.1)
Fetch FlightVolumes for cell 1x/2x/1y
FlightVolumes
Deconflict new reservation with ALL known flight volumes within cell.
PUT GridCellMetadata(1z/1x/1y, synctoken_v0(1.1.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v2(1.1.1)
PUT GridCellMetadata(1z/2x/1y, synctoken_v1(1.2.1), {uss_id}, {operation_endpoint},...)
Metadata Updated
200 Response and synctoken_v2(1.2.1)
Once all 200s are received, commit in USS A local DB for serving other USSs
Data Node API
USS A
USS B

**Use Case 4:** a USS updating multiple grid cells and another USS updates one of the cells during the update.


nterUSS Platform operation request concurrency protection
Data Node API
USS A
USS B
USS A Receives request to reserve airspace within cells 1z/1x/1y
GET GridCellMetadata(1z/1x/1y)
GET GridCellMetadata(1z/1x/1y)
Data is consistent, Metatdata is not yet updated, no way to contact A. If A is contacted, they do not serve the new operation until 200s from PUT.
Normal OCC process of retry on sync_token change handles race condition if B creates an operation
Empty Metadata and synctoken_v0
PUT GridCellMetadata(1z/1x/1y, synctoken_v0, {uss_id}, {operation_endpoint},...)
GET GridCellMetadata(1z/1x/1y)
Data is consistent, Metatdata not yet updated
letadata Updated
GET GridCellMetadata(1z/1x/1y)
Metatdata is now updated with USS A info. To cause an inconsistency, USS B has to process the response, AND contact USS A before the 200 is processed by USS A.
Metadata and synctoken_v1
Fetch FlightVolumes for cell 1z/1x/1y
FlightVolumes
This risk is extremely low, as the time between write complete and 200 is extremely minimal.
200 Response and synctoken_v1(1.1.1)
However, if USS A takes a long time to write to its own db, the risk increases.
Once all 200s are received, commit in USS A local DB for serving other USSs
Data Node API
USS A
USS B

**Use Case 5**: examples of the information concurrency protection built-in to the design. The risk of information being out of date or inconsistent is extremely low. Additional features can be added to the platform to eliminate this risk in the future, including a version, signature, or checksum in the metadata or adding a requirement for USSs to delay replying to specific operation requests during the PUT->write to local DB phase.

### Information Interface

In the information consistency node, the following rules are used for data storage:

- Data is located in the data store at /uss/gridcells/{zoom}/{x}/{y}/manifest.
- Manifest metadata includes monotonically increasing version number, timestamp, USS entries (including version number when updating and endpoints to request data from other USSs).
- The structure in JSON, which is extensible for additional data as required:

```
{version: <version>, timestamp: <last_updated>, operators:
    [{uss: <ussid>, scope: <used_for_obtaining_oauth_tokens>,
    version: <last_version_for_this_uss>,
    timestamp: <last_updated>,
    operation_endpoint:
         <web_service_to_retrieve_operations_in_this_grid>,
    operation_format: <output_format_of_uas_operations>,
    minimum_operation_timestamp:
         <lowest_start_time_of_operations_in_this_cell>,
    maximum_operation_timestamp:
         <highest_end_time_of_operations_in_this_cell>
    },
    …other USSs as appropriate… ]
}
```

The data structure consists of the following fields:

- **version:** the monotonically increasing version number, incremented on every change, including deletes;
- **timestamp:** the last date and time this grid cell has changed, updated on every change, including deletes;
- **operators:** a list of USSs with active operations in this grid cell, consisting of:
    - **uss** is the plain text identifier for the USS, as provided by OAuth;
    - **scope** is the web service scope to use to obtain OAuth token for this particular USS;
    - **version** is the version this USS last updated their entry, useful for identifying which USS changed when re-planning due to an update;
    - **timestamp** is the last date and time this grid cell has changed for this USS, updated on every change;
    - **operation_endpoint** is the submitting USS endpoint where all flights in this cell can be retrieved from;
    - **operation_format** is the output format the operation endpoint will return operations in (i.e. NASAv3, GUTMAv1);
    - **minimum_operation_timestamp & maximum_operation_timestamp** are the

time bounds of all of the USSs operations in this grid cell. This is used for quick filtering of conflicts by time and minimizing information sharing.

An example is below, stored at /uss/gridcells/15/16046/12355/manifest (Wing was the last to update, purposefully not in syntactically correct JSON format for easier reading):

```
{version: 112, timestamp: 2018-01-01T12:00:00.123Z, operators: [
 {uss: wing, scope: https://x.company/wing/interuss, version: 112, timestamp:
 2018-01-01T12:00:00.123Z, operation_endpoint: https://x.company/wing/15/16046/12355/ops,
 operation_format: GUTMAv1, minimum_operation_timestamp: 2018-01-01T12:01:00.123Z,
 maximum_operation_timestamp: 2018-01-01T12:06:00.123Z},
 {uss: anra, scope: https://anratechnologies.com/interuss, version: 87, timestamp:
 2018-01-01T02:20:40.244Z, operation_endpoint:
 https://anratechnologies.com/f?z=11&x=565&y=795, operation_format: NASAv3,
 minimum_operation_timestamp: 2018-01-01T18:00:00.123Z, maximum_operation_timestamp:
 2018-01-01T19:15:00.123Z},
 {uss: airmap, scope: https://airmap.io/scope/interuss, version: 110, timestamp:
 2018-01-01T11:59:00.000Z, operation_endpoint:
 https://airmap.io/flights/15/16046/12355/f, operation_format: GUTMAv1,
 minimum_operation_timestamp: 2018-01-01T17:45:00.123Z, maximum_operation_timestamp:
 2018-01-02T00:06:00.123Z}
]}
```

### Information Consistency (Zookeeper)

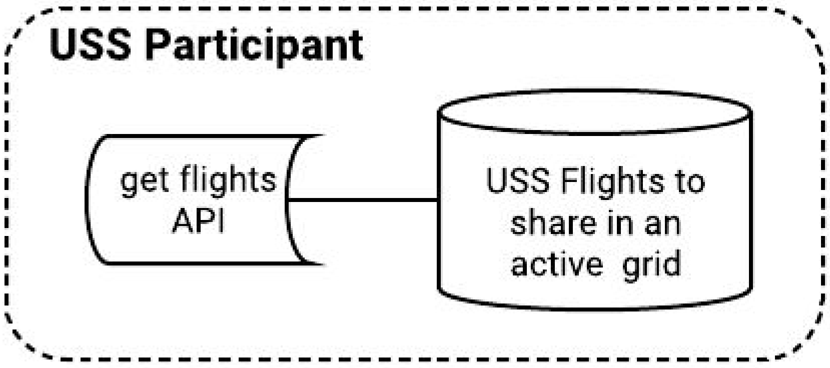
ZooKeeper messaging operates the similar way a classic two phase commit works. Proposals are sent to all ZooKeeper servers and committed when a quorum of them acknowledge the proposal.

A Message is a sequence of bytes to be atomically broadcast to all ZooKeeper servers. A message put into a Proposal and agreed upon before it is delivered. Total ordering is achieved using a ZooKeeper transaction id (zxid). All proposals are stamped with a zxid when it is proposed and exactly reflects the total ordering.

All requests that update ZooKeeper state are forwarded to the leader. The leader executes the request and broadcasts the change to the ZooKeeper state using an Zookeeper Atomic Broadcast (ZAB) protocol. The server that receives the client request responds to the client when it delivers the corresponding state change. ZAB uses by default simple majority quorums to decide on a proposal, so ZAB and thus ZooKeeper can only work if a majority of servers are correct. These parameters are adjustable and will be configured specifically for the USS distribution and requirements of the InterUSS Platform.

## USS Participant

A participant in the InterUSS platform has to, at a minimum, provide a specific grid cell API for other USSs to call to retrieve operations. This API should be protected with an access token, provided by and validated by the authentication node for that region.



### Required Web Service

In order to facilitate USS to USS communications, each USS is responsible for hosting a web service specified in **operation_endpoint**. This web service is hosted by the USS with an operation in the cell, which other USSs call to pull the operations within that specific grid cell.

The USS can provide their operations in any format approved by the InterUSS Platform Technical Steering Committee (currently NASAv3 and GUTMAv1).

- **NASAv3**: This implements GET /operations in the NASA USS v3 API specification[7].
- **GUTMAv1**: This implements flight declaration protocol in the Global UTM specification[8].

It is expected the grid cell specification (zoom, x, y) is in the endpoint address, for example GET /flights/{zoom}/{x}/{y}. This could also be hashed to prevent "guessing" at other cells without first interfacing with a data node. There are no other parameters needed, as the web service will return all current and future flights, with no limit on state or id. Special consideration should be taken by USSs for ensuring they provide all of the operation data via this endpoint as soon as the data node is updated.

### Authorization

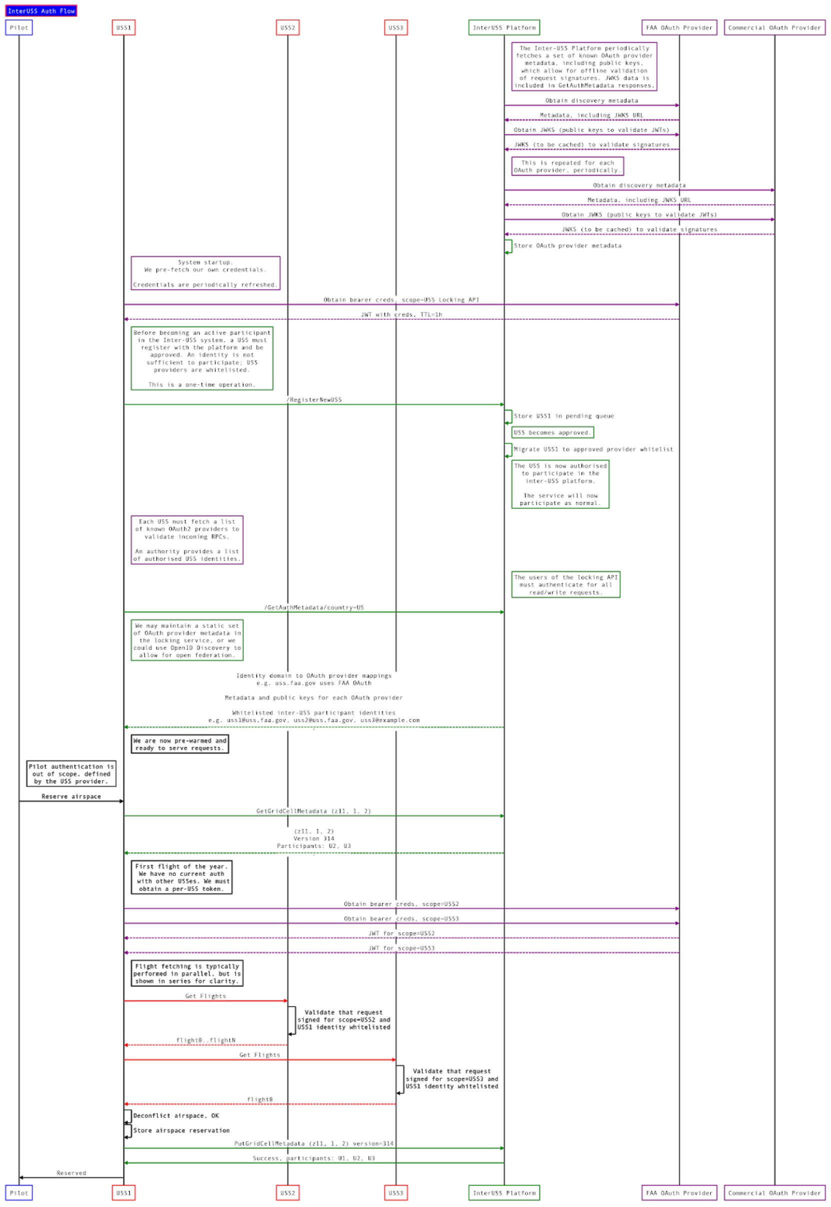
[7] https://app.swaggerhub.com/apis/utm/uss/v3#/A._Required_Endpoints/get_operations
[8] https://bitbucket.org/global_utm/flight-declaration-protocol/

TODO

## InterUSS Authentication Node

An authentication node implements standard OAuth capabilities, with scopes defined for approved USSs to access the data nodes and other USSs. This node may be provided by a government authority for certain regions, and by InterUSS members in regions not covered by an authority.

TODO


InterUSS Auth Flow
Pilot
USS1
USS2
USS3
InterUSS Platform
FAA OAuth Provider
Commercial OAuth Provider
The Inter-USS Platform periodically fetches a set of known OAuth provider metadata, including public keys, which allow for offline validation of request signatures. JWKS data is included in GetAuthMetadata responses.
Obtain discovery metadata
Metadata, including JWKS URL
Obtain JWKS (public keys to validate JWTs)
JWKS (to be cached) to validate signatures
This is repeated for each OAuth provider, periodically.
Obtain discovery metadata
Metadata, including JWKS URL
Obtain JWKS (public keys to validate JWTs)
JWKS (to be cached) to validate signatures
Store OAuth provider metadata
System startup. We pre-fetch our own credentials. Credentials are periodically refreshed.
Obtain bearer creds, scope=USS Locking API
JWT with creds, TTL=1h
Before becoming an active participant in the Inter-USS system, a USS must register with the platform and be approved. An identity is not sufficient to participate; USS providers are whitelisted. This is a one-time operation.
/RegisterNewUSS
Store USS1 in pending queue
USS becomes approved.
Migrate USS1 to approved provider whitelist
The USS is now authorised to participate in the inter-USS platform. The service will now participate as normal.
Each USS must fetch a list of known OAuth2 providers to validate incoming RPCs. An authority provides a list of authorised USS identities.
The users of the locking API must authenticate for all read/write requests.
/GetAuthMetadata/country=US
We may maintain a static set of OAuth provider metadata in the locking service, or we could use OpenID Discovery to allow for open federation.
Identity domain to OAuth provider mappings e.g. uss.faa.gov uses FAA OAuth
Metadata and public keys for each OAuth provider
Whitelisted inter-USS participant identities e.g. uss1@uss.faa.gov, uss2@uss.faa.gov, uss3@example.com
We are now pre-warmed and ready to serve requests.
Pilot authentication is out of scope, defined by the USS provider.
Reserve airspace
GetGridCellMetadata (z11, 1, 2)
(z11, 1, 2) Version 314 Participants: U2, U3
First flight of the year. We have no current auth with other USSes. We must obtain a per-USS token.
Obtain bearer creds, scope=USS2
Obtain bearer creds, scope=USS3
JWT for scope=USS2
JWT for scope=USS3
Flight fetching is typically performed in parallel, but is shown in series for clarity.
Get Flights
Validate that request signed for scope=USS2 and USS1 identity whitelisted
flight0..flightN
Get Flights
Validate that request signed for scope=USS3 and USS1 identity whitelisted
flight0
Deconflict airspace, OK
Store airspace reservation
PutGridCellMetadata (z11, 1, 2) version=314
Success, participants: U1, U2, U3
Reserved
Pilot
USS1
USS2
USS3
InterUSS Platform
FAA OAuth Provider
Commercial OAuth Provider