InterUSS design document

F3548-21 - User notification test

Version: 1.0
Status: Published
Editable file: Google doc

In the context of ASTM F3548-21 requirements testing, the InterUSS automated verification framework is required to evaluate if a notification has been dispatched to UAS personnel, end-users or operator's automation. The evaluation can include content of the notification and performance measurement of its delivery.
This document evaluates various approaches and proposes a common design to evaluate user notifications.

Analysis: Requirements of interest

The following table lists the requirements which depend on notifying UAS personnel, end-users or the operator's automation system. Requirements clarification questions and remarks can be found below.

Requirement of interest	***Notification trigger***	***Expected performance (seconds, 95% of the time)***	***Expected notification's content or "-" if nothing required.***	
GEN0400	*USS malfunction*	*UssFunctionFailureNotificationMax*	-	
GEN0405	*On all state transitions of operational intent*	*UserOiStateChangeNotificationMax*	-	
SCD0090	*Conflicting operational intent upon creation / modification*	*ConflictingOIMaxUserNotificationTime*	"reporting the conflict" [1]	
SCD0095	*Conflicting operational intent upon USS notification*	*ConflictingOIMaxUserNotificationTime*	"reporting the conflict" [1]	
ACM0010	*Detection of a period of nonconformance*	*period of time required by regulation (if applicable) or within MaxNonPerformanceNotificationLatency hours*	USS shall send a performance notification which is defined by ACM0015 (5.5.2.3): "A performance notification	

			shall (ACM0015) include, at a minimum, the period of time the performance notification addresses and the aggregate performance against each applicable conformance requirement (OPIN0005, OPIN0010).” [2]	
CMSA0115	*No position data is received when performing position report-based detection of nonconformance*	*OiMaxUpdateTimeNonconf*	-	
CMSA0300	*UA is outside its operational intent*	*OiMaxUpdateTimeNonconf*	-	
CSTP0005	Operational intent creation or modification by USS performing Constraint Processing	-	“providing the details of all constraints that intersect that operational intent” [3]	
CSTP0010	*Inability to provide details of all relevant constraints to UAS personnel or the operator’s automation system*	*IntersectingConstraintUserNotificationMax*	-	

CSTP0020	*Inability to provide details of all relevant constraints to UAS personnel or the operator’s automation system upon USS notification*	*IntersectingConstraintUserNotificationMax*	“USS shall (CSTP0020) send a user notification providing the details of the intersecting constraint” [3]	
CSTP0025	USS performing Constraint Processing creates a non-op-intent Subscription for Constraints	-	“USS shall (CSTP0025) notify the end user providing the details of all constraints that intersect with that area of interest.“ [3]	
CSTP0030	*Inability to provide details of constraints intersecting area of interest to end user*	*IntersectingConstraintUserNotificationMax*	-	
CSTP0035	*USS receives notification of Constraint update*	*IntersectingConstraintUserNotificationMax*	The details of the intersecting constraint	

[1] SCD0090 and SCD0095

The standard states that the USS shall send a notification **reporting the conflict**.

The check will be focused on notification detection by the means available in the test harness – example means could include “collecting outgoing notifications as they leave the USS bound for the client”, “collecting incoming notifications to the client app”, “performing OCR on the actual client app as observed by a physically-separate camera to detect visible notifications”, etc. It is the responsibility of the USS to justify to regulator the means of compliance of the USS’s implementation of the harness.

The recipient of notifications considered for these requirements should be the virtual user that planned the flight (the virtual user uss_qualifier actuates with the flight_planning interface).

Checks:
- The conflict has been notified to the user.
- The notification reported X conflicting operational intents.
- Notification was delivered within ConflictingOIMaxUserNotificationTime seconds, 95% of time.

[2] ACM0010

To be considered but not a priority in the medium term.

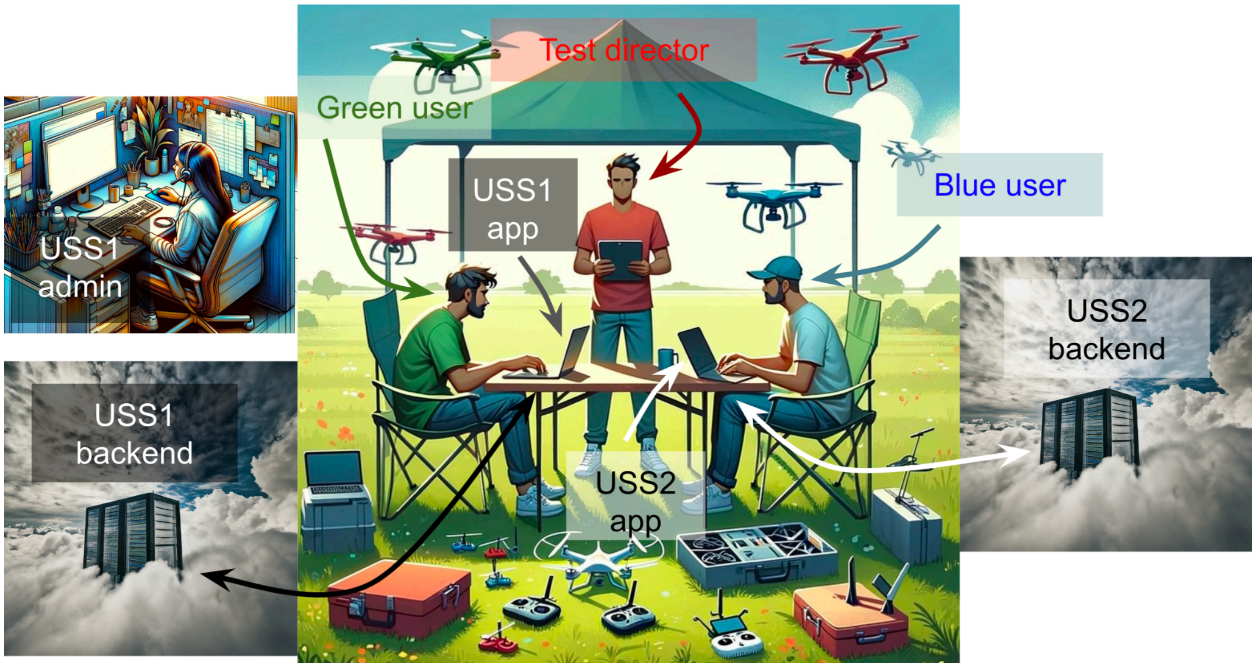
[3] CSTP0005, CSTP0020 and CSTP0025

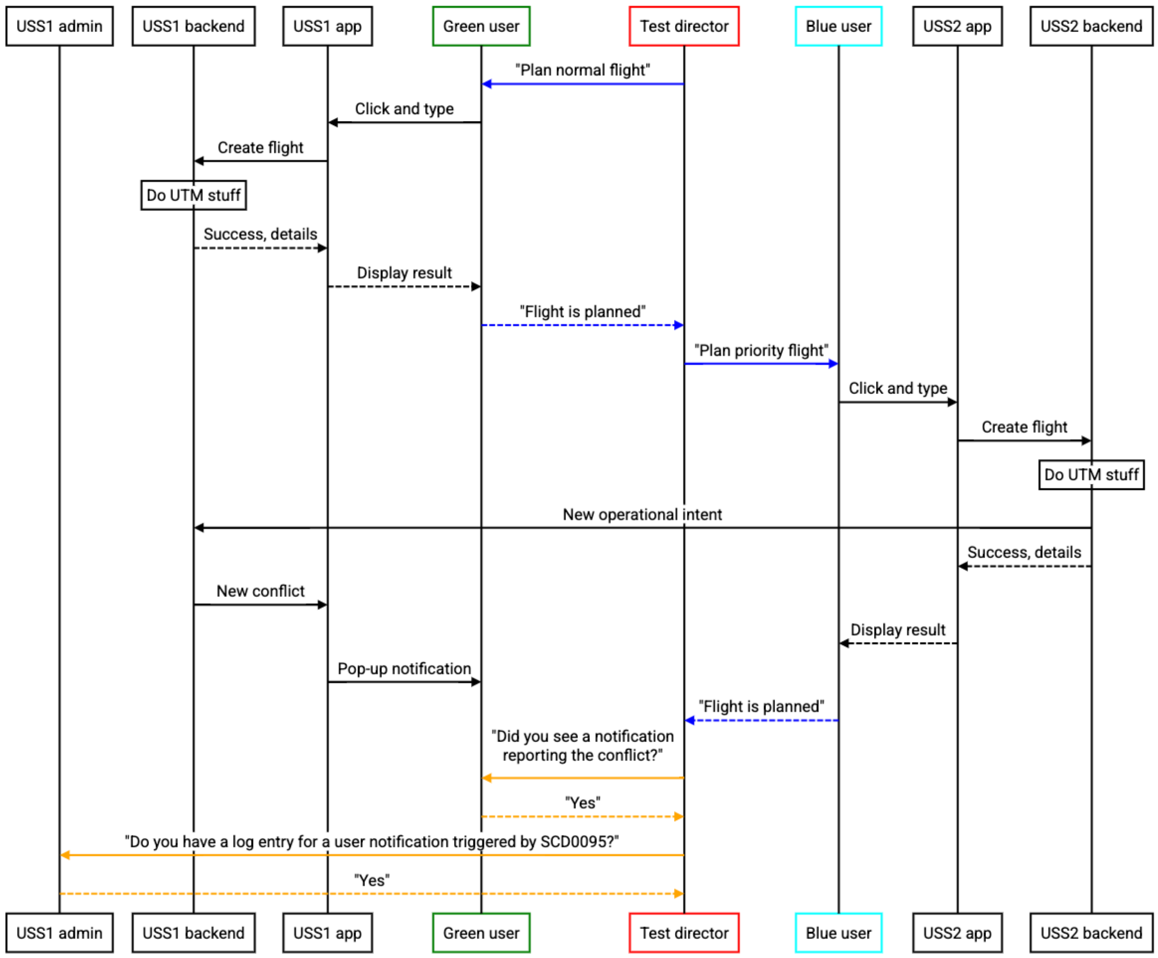
The standard requires the details of the intersecting constraint to be provided in the notification. An optional field will be required. Note that the schema of this field will be developed at a later stage.

Overall Testing Approach for SCD0095

Manual test

Before considering how to automate a test, it may be worthwhile to consider/visualize how an equivalent manual test would be performed. The below diagrams illustrate a manual UTM test scenario as might be conducted during a checkout (perhaps for a demonstration activity), illustrating the “happy path” (what we want to happen when everything is fully-compliant) and emphasizing the test director’s perspective.


Test director
Green user
USS1 app
Blue user
USS1 admin
USS2 backend
USS1 backend
USS2 app


USS1 admin
USS1 backend
USS1 app
Green user
Test director
Blue user
USS2 app
USS2 backend
"Plan normal flight"
Click and type
Create flight
Do UTM stuff
Success, details
Display result
"Flight is planned"
"Plan priority flight"
Click and type
Create flight
Do UTM stuff
New operational intent
Success, details
New conflict
Display result
Pop-up notification
"Flight is planned"
"Did you see a notification reporting the conflict?"
"Yes"
"Do you have a log entry for a user notification triggered by SCD0095?"
"Yes"
USS1 admin
USS1 backend
USS1 app
Green user
Test director
Blue user
USS2 app
USS2 backend

The test director makes the critical SCD0095 checks based on those last exchanges; they mark a green checkmark for USS1 if the Green user responds “Yes” and a red X if the Green user responds “No”, then a green checkmark for USS1 if the USS1 admin responds “Yes” and a red X if the USS1 admin responds “No”.
Note that either of the previously mentioned checks, if successful, would be sufficient to validate SCD0095 requirement. The second check would partially cover LOG0030 verification as well.

In the remainder of this document, we explore how a test similar to this could be automated with uss_qualifier serving as test director. For instance, the blue lines are equivalent to the flight_planning API each USS implements for virtual Green and Blue users. We don’t currently have automated testing APIs that allow the orange communication to happen, so we may need to add something in order to effectively test the requirements mentioned above.

Content evaluation

This section describes the schema of the minimal data required to check the requirement.

To verify compliance to SCD0095 based on the requirement analysis above, the automated testing interface (the automated equivalent to the orange exchange between the test director and green user above) will need to expose the following information regarding notifications to the user:

- *intersection_detected_at*: Timestamp of when the managing USS “becomes aware” of the intersection.
    a. In the case of SCD0090 (Conflict upon creation / modification), the timestamp of the flight planning response (OperationID: UpsertFlightPlan) will be used. It corresponds to the first “Flight is planned” (assuming it failed due to conflict) in the Manual test sequence diagram.
    b. In the case of SCD0095 (Conflict upon subscription notification), the USS subscription notification reply (204) is evidence of the USS awareness of the conflict and the timestamp of the response would be the start time for evaluating ConflictingOIMaxUserNotificationTime. In the Manual test sequence diagram, this corresponds to the response of the “New operational intent” assuming USS2 is the instrumented mock uss.
    Note that for testing SCD0080, the mock_uss is instrumented to capture the exchanges related to subscriptions between USSs. It is used to evaluate the delivery of notifications of operational intent changes. Therefore, by planning the conflicting operational intent with it, the USS under test will be notified and response to that notification can be recorded and exposed to the test driver.
- *notification_sent_at*: Timestamp of when the notification was sent by the USS.
- *conflicts_count* (optional): Number of conflicting operational intents reported by the notification. (Not needed CSTP0005, CSTP0020 and CSTP0025)
- *constraints* (optional): Details of the conflicting constraints. (Not needed for SCD0090 and SCD0095)

Data retrieval

This section evaluates options to expose the user notifications to the USS qualifier. Note that these options are not mutually-exclusive; options A and B could potentially both be implemented (though we would likely still want to understand which one should be prioritized for implementation first).

Option A: Expose the history of notifications in the Flight Planning API

This approach would extend the Flight Planning API with a new endpoint which allows the USS qualifier to retrieve information about the notifications received by the virtual USS user using polling. Following the equivalent manual test approach presented above, the test harness will be responsible to respond if it has seen a notification on behalf of a user.

The endpoint would need to be able to retrieve all notifications received by the user for a certain configurable period of time. Note that a test harness represents one designated representative USS user.

Endpoint specification proposal:
GET /user_notifications?from=[ISO_DATETIME]&to=[ISO_DATETIME]

Returns the list of notifications since the datetime provided in **'from'** argument until **'to'** argument by date in descending order. (**'to'** optional. Default: latest)

Upon dispatch of a notification to a user, the USS under test shall provide the notification information through the test harness in no more than 5 seconds. This information should be queryable until the end of the test run. This gives the USS under test some extra time to gather and process the notification information, which could require, for instance, parsing logs or aggregating records from various sources.

Working drafts:
- https://github.com/interuss/automated_testing_interfaces/pull/19
- https://docs.google.com/document/d/1zl3_UmFg6nWZ-crUNc9amKZFuGh2cDKi79nua64RmyA/edit

Pros:
- Simple to implement if the amount of information required to check the requirement is limited
- Can cover requirements outside the ASTM F3548-21 standard
- Information available at runtime

Cons:
- Complex if the amount of information required to check the requirement is large.

- Complex for USS to provide start time to compute ConflictingOIMaxUserNotificationTime through this endpoint. However, the computation approach proposed here would not require it.

Alternatives

Option B (PARKED): Export logs from LOG0030 post run

LOG0030 requires a USS to log according to a standard format all instances of interactions with UAS personnel, end users, or the operator's automation.
No interface to retrieve the data is mandated, however the ability to export in such format is specified in Annex III:
https://redocly.github.io/redoc/?url=https://github.com/astm-utm/Protocol/raw/v1.0.0/utm.yaml#tag/Logging/operation/getLogSet

Evaluation of the logs may consist of:

- The automated testing interface could be extended to allow the USS qualifier to export those logs asynchronously. A new scenario could be added at the end of the test suite to request the log preparation, download and evaluate them.
- A new test suite exclusively evaluating the logs to be run offline could be provided as an alternative means of evaluation. The report of the test session may be required as input to compare log results with USS qualifier records.

Pros:

- Close to F3548-21
- All data available, therefore suitable if the amount of information required to check the requirement is large.
- Would validate LOG0030 requirement
- Information should already be stored in some way as per 5.9.1.2.
- Approach can be used with other LOG00X requirements

Cons:

- There is no requirement for USSs to have the logs available "online". Indeed, we could imagine a USS service using a complex aggregation job run manually upon request. This process may not be executed in a reasonable timeframe for a test session. That could require an extensive amount of logic to automate the log extraction specifically for the tests, which is in principle undesirable.
- Limited to F3548-21 requirements

Option C (REJECTED): Notification receiver server

USS qualifier or any other controlled system exposes an interface which plays the role of the operator
Cons:

- Requires synchronous notification between USS under test and the USS qualifier which increases the complexity of the implementation, may impact the performance of the test and impact the results of the tests.
- uss_qualifier does not host any endpoints currently, so this introduces a new level of complexity and maintenance. Even if it's tacked onto mock_uss (which it logically shouldn't be – receiving notifications isn't something a USS would do; rather this is a piece of test infrastructure)

Performance evaluation

Option B: Statistical approach

For SCD0085, a statistical approach has been adopted in order to reasonably evaluate the performance of a response with a limited number of data points.

Pros:
- Uses a similar pattern in the code base.
- Short in terms of test suite duration.

Cons:
- Limited samples

Alternatives considered

Option A (PARKED): Load test

Create a significant amount of notification occurrences and aggregate the results.

Pros:
- Test the system under load.

Cons:
- Increase significantly the duration of the test suite.

Discussions

Once discussions are resolved, they are moved to a subsection here where they can be preserved for future reference.

General remarks

[Hrishi] An interesting project that some of you know of https://github.com/dronecode/utm-adapter, this standardizes how a GCS interacts with UTM providers, it would be useful to integrate these notification mechanisms as a reference implementation there.

> [Michael] Thank you Hrishi for the input. I don't see any interface related to user notifications in the ussp api spec on the master branch. Is there another location I should look at? Are you aware of any standardisation body who may have produced a normative specification on that topic?
>
> [Hrishi] We use a AMQP channel to notify the QGCS (there is a active PR in QGCS repository for this), in that way there is REST API spec, I can discuss with the group if there should be a REST based API for notifications. Also I know there is a WG within EUROCAE who is interested in working and I am presenting the Dronecode adapter work to them this week. We can talk about this tomorrow perhaps? I have been travelling so could not join the interuss weekly calls. I plan to join from tomorrow.
>
> [Hrishi] Thank you for raising this. The expectation at this point is that if we wanted to verify compliance to those standards using the InterUSS test suite framework, the automated test interface used would be the flight planning and specifics of each standards would be addressed by the test harness implementation. We definitely want the flight planning automated test interface to support USS implementation of standard relying on other protocols than HTTP. So, feel free to review the outcome of this work and raise any concern.

Comment on a green checkmark for USS1 if the USS1 admin responds “Yes” and a red X if the USS1 admin responds “No” :

[Seungman] Is this check for verifying SCD0095 or for verifying LOG0030 requirement? I think this check is more related to LOG0030, rather than SCD0095. Isn't the first check (the first two exchanges) enough for SCD0095?

> [Michael] That's my understanding as well. This assumption allows us to validate SCD0095 requirement with Data retrieval Option A without implementing Option B for now (for this specific requirement).

Identifying the recipient of a notification

In response to “Is it enough to test the presence of the notification?”

[Julien] Depending on how we intend to obtain the notifications, we might also need to know to whom the notification has been sent to?

The situation I'm thinking of: if we're using the approach of obtaining notifications via logs after the qualifier has run, we may end up with a bunch of notifications that can not clearly be mapped to the event that caused them and/or the operator they were sent to.

Ie, whatever we ask the USS's to provide must make it reasonably easy to determine what was the cause of the notification and to whom it was intended.

> [Ben] I think it's true a notification sent to any random user cannot be used as evidence that a notification was sent to the required user (the operator/personnel associated with the op intent). However, I think we can potentially make this easy by defining the endpoint to get user notifications (or whatever equivalent we design) to get notifications only for the user associated with the flight. We're already assuming a particular user in the flight_planning API because the flight_planning API is all about puppeteering that particular user. If a test needs two separate users for the same USS, I would expect that to be accomplished by providing two instances of the flight_planning API. A USS could generalize this by, for instance, making the base URL of their flight_planning API be something like https://autotesting.uss.example.com/flight_planning/{user_id}

In response to "Is the successful notification submission enough from the USS point of view ?"

[Ben] For SCD0095, I think successful submission of a notification regarding a conflict is enough (if the notification doesn't report the conflict, it isn't sufficient).

From "Content Evaluation" under "Overall Testing Approach for SCD0095", the following field was considered:

1. recipient: The user to which the notification was sent
    - [Julien] Should we include to whom the notification was sent, as well, assuming there is some form of generic id that we know is available and that we can recognize as being the one used by the qualifier?
        i. [Ben] I think that logical information certainly has to be present, but I think we can gather it via inference rather than needing it to be an explicit field in the API.
            1. [Mickael] Agreed with inferring those info, that is consistent with other APIs, and doable since we assume a lock on areas that are subject to tests.

From "Content evaluation", considering an element that may need to be exposed for a notification:

2. Recipient information ?
    - [Joey] I don't understand the "user information"? What does this reference in regard to the requirement?
    - [Michael] To clarify, I renamed it to recipient information. The question is, do we need to assess the actual recipient of the notification ? If yes, how ?
    - [Julien] My comment above probably applies here: I'd say that yes, ideally we have a way of finding out for who or what system a notification was intended for
    - [Ben] I'll suggest we continue one of the earlier discussions rather than this one

1. sent_at renamed to detected_at: Timestamp of when the notification was detected
    - [Ben] I think we probably want to make this a little more generic; e.g., “Timestamp of when the notification was detected” – that enables many different implementations for detecting the notification rather than one dependent on the user (which we couldn’t actually determine in an automated system any way as a human user wouldn’t receive the notification until they actually looked at the screen)
        i. [Michael] I think we reached an agreement here, I will rename sent_at to detected_at.

Identification of the conflict

*From “Content Evaluation”, in response to “*Should the conflict be identified ?”

> [Ben] The requirement is that the notification must report the conflict. It seems like not any user notification suffices to meet this requirement as it must report the conflict and not all notifications report the conflict – so, I think we would need to be able to determine that the notification is in regard to a conflict (“identify the conflict”). However, I don’t think this establishes a requirement to, e.g., differentiate between conflict A and conflict B (“identify the conflict”). One way to meet these needs could be to have an is_regarding_conflict field on a UserNotification object.
>
> > [Joey] Since a conflict is always 1:1, we should be able to just have the USS note the two things that are in conflict by a reasonable identifier. For example if we are alerting an operator of a new constraint conflict with a planned operation, then the USS would know the operation's id and the constraint's id. That pair of IDs would suffice for us to know that the alert was about that conflict. I don't think a boolean alone is sufficient. that would set the content, but doesn't answer the "did the right user/system" get the notification.
> >
> > > [Punam] I think we might need one more info, that is the ovn to exactly identify the current state of conflicting operation or constraint. As there could have been modifications to the same operation id or constraint id.
> > >
> > > [Ben] In the equivalent Manual test below, is USS1’s app required by ASTM F3548-21 to include the operational intent ID in the notification that the Green user sees? If not, and the USS did not include the operational intent ID in that notification, would the test director be justified in marking a red X (indicating they determined that USS1 did not comply with SCD0095) when the Green user said there was no operational intent ID included in the notification? Same questions but for OVNs re: Punam.

Re: did the right user get the notification, in the manual test below, the test director is asking the Green user if they saw a notification. The Green user can't see notifications for other users (or, if they could, those could be treated as notifications to the Green user since they saw them).

[Michael] In the case of SCD0090 and SCD0095 (in opposition to CSTP0005 for instance), my understanding is that we are evaluating the signal of the change. I would like to propose in order to move forward to limit the check to the detection of the conflict without being specific about it. If I relate to the experience on another project, the notification would arrive on a smartphone including only the text "Conflict detected. Open the application for details.". Retrieving the details would be achieved in a second step. So, can we agree to target the lowest common denominator featureset? Asking for details of the conflict (for this specific requirement) may go beyond that and require adding complex logic to the USS test harness for this specific test. Designing a scenario such that the conflict is well-known should be sufficient. I am obviously open to revisiting that approach if we identify limitations while implementing or if we find an established requirement in the standard which requires it.

[Call on Feb 6 with Ben, Joey, Punam, Seungman and Michael] It was agreed to proceed without requiring the test harness to identify specifically the operational intent details of the conflicts. However, it was pointed out that it may not be not enough to only evaluate the presence of a conflict. Therefore, the solution retained is to require the number of conflicting operational intents reported by the notification.

from "Content evaluation", regarding "is_regarding_conflict: Boolean indicating whether the notification is regarding a conflict":

- [Ben] I think this is sufficient to verify SCD0095 and should be available via inspection of a notification, regardless of how it is presented to the user
- [Michael] As agreed, the number of conflicting operational intent is preferred to develop more complex scenarios.

from "Content evaluation", regarding "Conflict information ?"

- [Joey] Suggest the operational details as known by the USS managing that conflicting operation is the appropriate information about any given conflict. The client system can do what it deems appropriate with that information (per the established user agreement between USS and operator).
  i. [Ben] Is there a requirement that the USS provide that information to the Green user? If not, and the test director asks the Green user for that information and the Green user responds that they don't have that information, what should the test director do?
- [Ben] ~~I'm not clear on the difference between "Notification metadata" and "Notification information", but~~ I would expect an optional field to be populated with OperationalIntentDetails and an optional field to be populated with

ConstraintDetails when the notification is fulfilling the CSTP requirements involving delivery of entity details (but not needed for checking SCD0095)

- [Michael] This will be impacted by the outcome of the previous section:
  i. 1 SCD0090 and SCD0095
  ii. 3 CSTP0005, CSTP0020 and CSTP0025
  Please comment there.

  [Michael] As agreed, only the aggregated number of conflicting operational intent will be required. An optional field will be required to provide CSTP* constraint information.

from "Content Evaluation", regarding "conflict_known_by_uss_at" renamed to "received_at"

- [Ben] This information is probably not present in the notification so we can't ask for it here if we are retrieving information about actual notifications – remember that one (exaggerated) implementation a USS could use to automate reporting of user notifications could be to have a camera pointed at the actual USS app and detect and OCR notifications. In that case, there would be nowhere to find this timestamp. Instead, we can determine a bounding value for this timestamp through other means – for instance, if the mock_uss was the USS delivering the notification, mock_uss's receipt of a 204 is a safe point where we can say that the managing USS has become aware of the conflict.
  i. [Punam] I think while testing astm standard requirements, we can make an assumption that we are asking for conflict detected based on astm data exchange. And we can state that the field is astm-specific in the field description.
  However, I do feel that in the astm standard rather than measuring ConflictingOIMaxUserNotificationTime against 'Time Operational intent conflicts identified by USS' it should be measured with time start as 'Time when the conflicting Operational intent was received by the USS'. That would then help to include the time a USS took to evaluate whether a conflict existed, in ConflictingOIMaxUserNotificationTime.

[Michael] Based on your inputs, it seems that we are in agreement with the two following approaches in order to set the start time when the managing USS becomes aware of the conflict:

- SCD0090 (Conflict upon creation / modification): The timestamp of the flight planning response (OperationID: UpsertFlightPlan). It corresponds to the first "Flight is planned" in the Manual test sequence diagram.
- SCD0095 (Conflict upon subscription notification): For testing SCD0080, the mock_uss is instrumented to capture the exchanges related to subscriptions between USSs. It is used to evaluate the delivery of notifications of operational intent changes. Therefore, the USS subscription notification reply (204) is evidence of the USS awareness of the conflict and the timestamp of the response would be the start time for evaluating ConflictingOIMaxUserNotificationTime. In the Manual test sequence diagram, this

corresponds to the response of the “New operational intent” assuming USS2 is the instrumented mock uss..

From “Content Evaluation”, Response of the client system? This is bonus metadata, might not be needed for testing purposes, but perhaps it can exist in a metadata schema?

- [Ben] Could you elaborate on what this might look like (data type, potential values, etc)? I’m not visualizing where I would get this information if I were a user looking at a notification in the USS’s app
    i. [Joey] For an HTTP-based system, it might be something like a 2XX or 4XX response from the client system to give a sense of success/failure. A USS operator would have these response values available and may show it in an "app" like any other system monitoring display (hey, the connection to client X is 'red'!). In other notification approaches (phone calls, taps on shoulders), it would clearly need to be something else.

       I might not be following your "user looking at an app" example because I wasn't clear on which "user" was meant. If it was an end user (pilot, UAS operator, etc.) looking at data from the USS then the USS would have successfully sent those data to the app somehow. If the user is a maintainer/operator of the USS system, that user would see connection status and alerts of failed HTTP calls or something.
        1. [Ben] I’ve added a description of an equivalent manual test above and in that case, I’m thinking of the Green user answering a question from the test director. Additional fields here are basically the test director saying “tell more more about the notification you saw”. I’m not sure what the Green user would say in response that would constitute “response of the client system”. sent_at above would be the Green user telling the test director when they saw the notification. is_regarding_conflict would be the Green user telling the test director the notification indicated a conflict.
- [Michael] Taking as a key assumption the parallel with the manual testing and trying to limit to the minimum the information asked to USSs, it seems that we would not need those additional metadata to evaluate this specific requirement. Can we agree to park it for now and bring it back if needed?

Data retrieval - Option A: Expose the history of notifications in the Flight Planning API

In response to “All notifications ? By flight plan ? By user ?”

[Julien] Could we do something like "all notifications caused by or related to entities owned by the caller"?

Ie, something that combines both:

- all notifications that a user caused through their actions (ie, notifications to other uses)
- all notifications that we would receive as a real user interacting with the system (ie, we created an op-intent, and some other action causes a conflict

In other words, for a given flight plan, there are "outbound" notifications (which are caused by actions of the owner of the plan), as well as inbound notifications (caused by actions by other parties in the system and which have an impact on the owner's plan).

Possibly we don't need access to both kinds?

> [Ben] I'm picturing translation of our design into a manual checkout (now, see Manual test above) and I'm not sure how I, as the designated representative USS user (Green user above) for the test (whom the test director commands to actuate the system and questions to determine system behavior), could report the difference between those two categories when telling the test director what notifications I saw.
>
> Since we don't currently attempt to differentiate between users when planning flights in the flight_planning and legacy SCD injection APIs (we assume there is just one designated representative USS user doing the planning), I think we can mirror that design here. Just as the flight_planning endpoints let the test director instruct the representative user to click stuff in the USS's app to perform flight planning activities, I would expect this endpoint to let the test director ask the representative user what notifications they received, and sometimes some additional information about those notifications.

Discussion on "Pros":

- Information available at runtime
  - [Punam] Can help with testing user notifications with the statistical approach, as mentioned under the Performance Evaluation section, as this user notification will be accessible during the test run.

Discussion on "Cons":

- Complex if the amount of information required to check the requirement is large:
  - [Julien] I'd also add that this adds some constraints to USSs: depending how they implement their test/injection harness, exposing notifications might be of variable difficulty

    Another factor is what kind of performance we can reasonably expect of such endpoints.
    - [Ben] I think we'd need to structure the data requested such that it should be easy for a USS to populate that data from the

information available in the notification regardless of where the USS decided to capture and expose that notification. That means we shouldn't ask for notification information that we don't know should be available to the user pursuant to some requirement.

I would expect pretty good performance – we're just talking about logging a minimal amount of information upon a human-scale interaction and being able to later query that information (likely based on timestamp).

- [Punam] USSes might not be happy about providing the timestamps for calculating ConflictingOIMaxUserNotificationTime through this interface, which they might be ok to give through the /getLogset endpoint as per UTM spec. To calculate ConflictingOIMaxUserNotificationTime as per the ASTM standard we need to know time start - Operational intent conflicts identified by USS and time end - Notification dispatched to UAS personnel.
    - [Ben] We do need to have information about the start time in order to measure a time period to compare against ConflictingOIMaxUserNotificationTime, but that information does not all need to be provided in this interface. In the Manual test above, the Green user may have no idea when the USS identified the conflict and therefore could not provide this information to the test director along the orange lines. Instead, the test director knows that USS1 must have identified the conflict no later than the time the Blue user said "Flight is planned". Therefore if there is no notification by ConflictingOIMaxUserNotificationTime (modified appropriately to account for 95%) after the Blue user said "Flight is planned", the test director can confidently write a red X next to the requirement even though the Green user never provided the timestamp for "Notification dispatched to UAS personnel".
        - [Punam] I do agree that it is difficult to know the start time as defined in the standard.
          Probably, the standard should instead ask for a time start that is easier for test driver to extract from the interactions in the test. Eg. when was the conflicting entity notified to the USS1 backend.
          If as test driver, we are unable to get the exact time, then should we state that we cannot test this requirement exactly as per the standard mentions? And, then measure it against the closest available time to the tests driver.

            [Ben] Standards development should generally happen where the appropriate stakeholders are present and engaged, so I think "the standard should" is a little strong without knowing the reason it was chosen to be the way it is. But regardless, InterUSS automated testing verifies compliance to externally-defined requirements, so we need to take them as given in any case for this project.

There is usually no way to get any particular “exact time” – for instance, a message could be “received” when it reaches a system’s external load balancer, or when it is enqueued for handling by the server daemon, or when the worker process dequeues the message to handle it, or at the entrypoint to the function handling the message, or when a log statement within the handler was executed. These are all different times for when the message was “received”. But that’s fine as we can still test just fine by being slightly conservative. For instance, we know a message was received before the time the receiver responded. Therefore, if there is a maximum duration starting with “received”, we can be very sure that requirement is violated if the duration is exceeded even assuming “received” started at the time of response.

In general, we can never test a requirement “exactly”. We can create situations where some types of non-compliance can be detected, and we can continually endeavor to augment these situations to capture as much of the expected types of non-compliance as practical. However, we can never test all possible situations, and so therefore we can never positively establish full compliance to a requirement (this is the same as any kind of checkout or demonstration-based verification). We want very high specificity with regard to detecting non-compliance, but the difference in sensitivity between detecting noncompliance at 5.05 seconds (against a 5 second requirement) and detecting noncompliance at 5.5 seconds usually does not have much value.

[Michael] The approach proposed here would not require to use this endpoint to retrieve the start information and calculate ConflictingOIMaxUserNotificationTime by the USS. Let’s evaluate that and revisit it if needed.

Data retrieval - Option B: Export logs from LOG0030 post run

- [Julien] Something that we might want to add to the "cons" of this approach: it will result in longer feedback cycles.

If the prober can check everything in a straightforward manner USSes max fix their problems faster.

I don't know how important this is in comparison to imposing an additional API with option A, but it's worth keeping that in mind.

- [Ben] I'm not a fan of two separate runs as I think it introduces a lot of additional conceptual complexity and/or breaks a lot of existing paradigms – instead, I would expect a test run to make the log export request and then sit and periodically poll status until exports were complete, and only then complete the test run. That way, there would only be one kind of test run and that test run would contain all of the necessary tests. If we really need to optimize, I would imagine we could be clever about putting all the log-generating scenarios near the front of the test and perform non-log-generating scenarios at the end after initiating the export for logs for the front of the test. But, while it's certainly true that there is potentially an absurd and unmanageable amount of information potentially available via these logs, our requests would be limited to a very specific time and place which should result in a very limited amount of data which should be fast and easy to export.
  [Michael] Looks like we are in agreement on a simple version which can be supported by Option A knowing that the code base is already geared to welcome that approach. I propose to start with that option and park this one until we tackle the LOG* requirements. Checks may be added later if we want to strengthen the coverage

Performance Evaluation

[Punam] This is good for tests where we want to verify that user notification was sent by a USS within a long enough

[Michael] Currently the test framework uses the statistical approach to assess other requirements (eg SCD0085). Load tests will certainly be added at some point to the framework but will require extensive design. I suggest that we start with Option B and address Option A in the long term jointly with other requirements.