

Devils in the Detail



In this, the 4th and final episode of a series of articles exploring the principles of ontological modelling, the Ontology Working Group (OWG) builds on previous episodes, introducing some new terms from the Unified Foundation Ontology (UFO) and expanding further on the worked example relating to flights and flight planning.

With these articles, the OWG aims to make the topic more accessible through a series of connected newsletter articles. Each episode expands on ontological aspects and, by the end, will hopefully provide the reader with enough knowledge to read and contribute to models in the future.

What Is an Ontology?

As we mentioned in earlier articles, an ontology is not some esoteric formalisation, which is nice to have, but is essential to people's day-to-day work on real systems.

When we build a system, we are modelling our world and this involves developing concepts, their relationships and the rules for connecting them – this is called an ontology, or more formally: **“an explicit specification of a conceptualization”** [1].

If not explicitly specified, establishing a shared conceptualisation, that is an understanding of interrelated concepts, will be much harder and miscommunication may be more common. Such shared understanding helps ensure that models used to support parts of the lifecycle of a system always remain coherent and consistent. If the concepts in the model are not defined in writing, meaning can just be held in the minds of the designers and be lost when they are no longer involved. An ontology serves as a contract of meaning between stakeholders, who make what are known as 'ontological commitments'. That is, they commit to produce engineering artefacts in a way that is consistent with the ontology that captures their shared conceptualization.

Where Did We Get To? – A Recap

We are now on our 4th episode ([1st in Feb 2025](#), [2nd in Oct 2025](#) and [3rd in Feb 2026](#)). This article should still be of interest if you have not read the previous three, but you will get a lot more value out of it if you read them first.

To recap on the earlier articles, the OWG uses the **Unified Foundation Ontology (UFO)** [2]. This is a foundation for ontological modelling implemented in **OntoUML** [3] (a customisation of the Unified Modelling Language, UML [4], with associated software support via application plugins [5]). The following notation was previously introduced.

- We use boxes and connecting lines (more formally **Classes** and **Relationships** using the modelling terminology, which are highlighted in this section in **bold**) to represent concepts in our worldview. For readability, specific **Classes** mentioned in the text of this article are shown in **bold purple**.
- Ontological aspects are expressed as **Stereotypes** (the part of the **Class** definition with Guillemets like this: «**role**»). **Stereotypes** mentioned in the text in this article are in **bold blue**.
- Arrows with a closed, unfilled triangle $\rightarrow$ are **Generalizations**, or in other words, one 'thing' subtypes the more general 'thing' that is being pointed to (for example, a **Cessna** is a subtype of **Aircraft** and **Aircraft** is a **Generalization** of **Cessna**)
- An open arrow $\Rightarrow$ shows an **Association** between things. For example, in the phrase "*Pilot 'flies' Aircraft*", the concept 'flies' would be modelled as an **Association**. Note that **Associations** also have **Stereotypes**.
- **Association ends** carry numerical expressions at the end of each association, such as **0..*** and **1..*** (read as *zero or more* and *one or more* respectively). These show **Multiplicity** – the valid range of the number of each type of thing that can be related. This is used to help express constraints, such as a **Pilot** can be trained to fly one or more **Aircraft**, but a given **Aircraft** can usually be flown by, at most, two **Pilots** at any one time.

We covered the important concept of identity principles: these form sets of characteristics that, in combination, give us the ability to separate one individual entity from another (individuate, more formally).

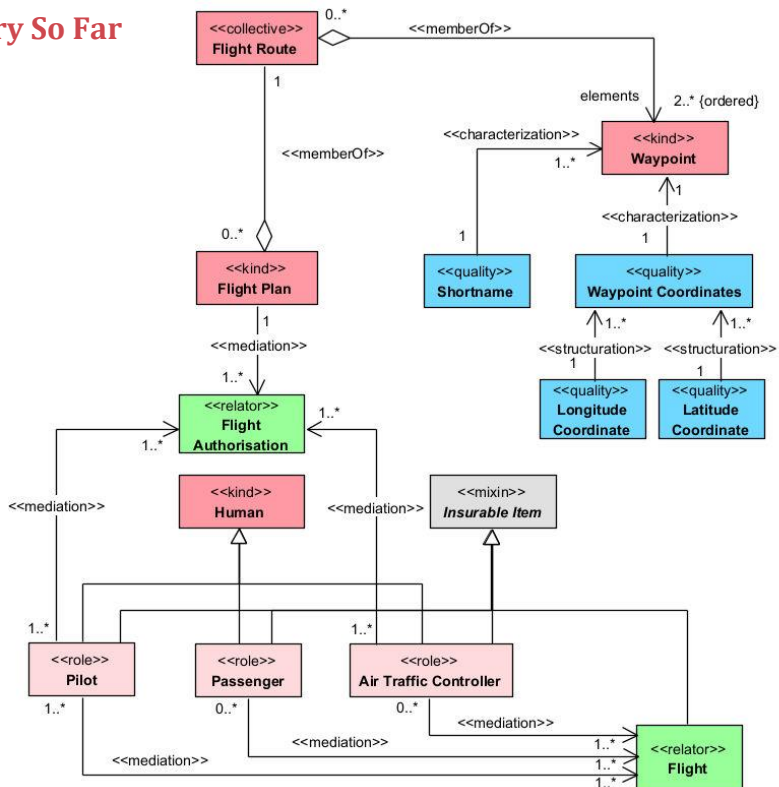
We have also covered some core UFO concepts, expressed as **Stereotypes**:

- **Kinds** reflect the objects we find around us like people, aircraft and dogs that share a common identity principle. **Subkinds** allow us to build classic taxonomies.
- **Roles** and **Phases** are types whose instances may be only temporary while the thing to which they applied still exists. The same **Human** may hold the **Roles Pilot** and **Passenger** and go through **Phases of Employed** and **Unemployed** at different times during their lifetime. We will explore in this article how logical rules may be needed to prevent conflicts in models that use these concepts.
- **Mixins** are used to describe shared properties of objects that do not all belong to the same **Type** and so can span multiple **Types** with differing identity principles.

In the last edition we also introduced the higher order categorisations of **Sortals**: things we can individuate and count (like **Pilots** and **Waypoints**) and things we can't, like **Loud**, **Metal** and **Fatigue** which are **NonSortals**. We also discussed rigidity: **Rigid Classes** like an **Aircraft** which is an **Aircraft** throughout its entire life as opposed to **AntiRigid** like an **Airborne Aircraft**, which exists only for part of the **Aircraft's** lifetime.

We finally expanded on an example based on **Flight Plans** and **Waypoints** to introduce the idea of a **Collective** of **Waypoints** forming the **Flight Route** and how **Quality** can be used with the stereotyped **Association Characterization** to describe what individuates a **Waypoint**. We also introduced **Structuration** to decompose different aspects of a **Quality**

Our Story So Far



We've developed our story over the last three episodes to arrive at the previous figure.

In the diagram, a **Flight Route** is a **Collective** associated with a number of **Waypoints** which are representations of physical locations on the Earth the **Pilot** intends to navigate between.

Waypoints have exactly one **Waypoint Coordinates**, which is modelled as **Latitude Coordinate** and **Longitude Coordinate**. Coordinates are alphanumeric values representing physical coordinate points on the Earth. For example, for the **Waypoint** with **Shortname ETOXU**, the **Latitude Coordinate** might have the representation: **N47 43.6** and **Longitude Coordinate: E009 33.0**

At the end of the last episode, we left the reader to consider a couple of questions that explore how robust our model is at the moment:

- In reality, an individual **Human** cannot be both a **Pilot** and a **Passenger** on the same **Flight** – does the model prohibit this situation?
- Does the model prohibit **Waypoints** located at different **Waypoint Coordinates** from having the same **Shortname**? If not, do you see any problems that might arise?

Let's explore the answers to these questions next.

A Crisis of Identity

The lower part of our model shows that **Pilot**, **Passenger** and **Air Traffic Controller** are roles of **Human** and shows their relationships to a **Flight**. The **Multiplicity** tells us a great deal about the relationships. A **Flight** must have at least one **Pilot** (1..*), it does not always need to have an **Air Traffic Controller** (0..*), although it might have several, and it does not necessarily need to have a **Passenger** (0..*), although it might have many. It also does not prevent a **Pilot** from also being a **Passenger** on the same **Flight**; so the model is under-constrained; it is modelling things as being independent when there are in fact relationships between them not reflected in the model.

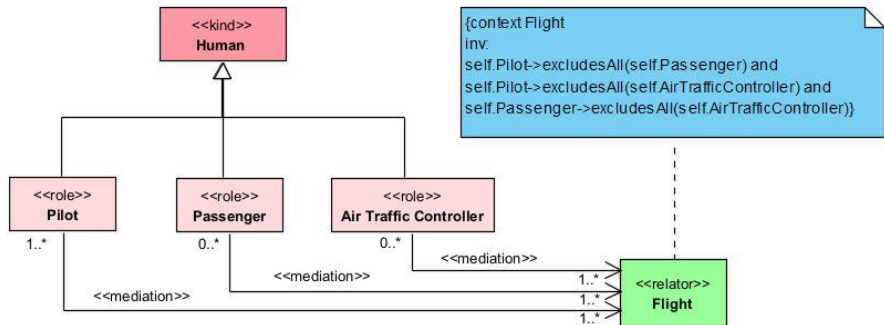
This is an example of a modelling pattern that designers may gravitate towards but in fact has pitfalls and so is classed as an *anti-pattern*. In this particular case, the "anti-pattern" is called the "Relator Mediating Overlapping Types" (or RelOver for short) [6]. (This is just one of many anti-patterns; see [9] for more examples).

This illustrates the wonderful power of ontologies; the defined **Classes**, relationships and rules ensure meaning can be stated in a way that helps reduce bias, preconceptions and implicit assumptions. In this case we see that while a model may be well-formed according to the language used to create the model, it may not precisely model the real-world situation you intended!

"This illustrates the wonderful power of ontologies; through their preciseness; the model defines the classes and their relationships so meaning can be stated in a way that helps reduce bias, preconceptions and implicit assumptions"

Clearly, we want our model to prevent illogical combinations, so we can add a constraint to indicate this in the model.

Many languages can be used to express constraints, the one we will explore is packaged within the UML specification and is called the Object Constraint Language (OCL). This has its own syntax, which we won't have time to explore in full in this article, but suffice to say it allows for more code-like specification of custom constraints. See the following diagram (other model elements have been omitted for clarity).



In this diagram we have added an OCL constraint on the **Flight** relator:

```

context Flight
inv:
self.Pilot->excludesAll(self.Passenger) and
self.Pilot->excludesAll(self.AirTrafficController) and
self.Passenger->excludesAll(self.AirTrafficController)
  
```

The notation `self.` refers here to the current instance of **Flight** being considered when the constraint is evaluated. For example, the real **Humans** that took the **Flight** denoted by “French Bee flight BF731 between Los Angeles and Paris Orly on the 28th Dec 2023”. Note that it is the set of **Humans** because that is common **Kind** shared by all of the **Roles** that are subject to the constraint (indicated by the **Generalization**).

`excludesAll(...)` is an operation that ensures, within the set of real **Humans** associated with a given **Flight**, that no single individual can simultaneously belong to both the source set (to left of the `->`) and the target set (inside the `(...)`).

The `and` operator serves its usual function in logic.

So the above constraint code is saying that, in the `context` of a **Flight**, we have an invariant (`inv`) constraint that ensures, for any given instance of **Flight** that:

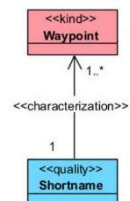
- A **Pilot** cannot also be a **Passenger** on the same **Flight**
- A **Pilot** cannot be an **Air Traffic Controller** on the same **Flight**
- A **Passenger** cannot be an **Air Traffic Controller** on the same **Flight**

Because the rules apply in both directions, the three elements of the constraint are sufficient to cover all possible pairings of **Pilot**, **Air Traffic Controller** and **Passenger**.

Note that the constraint only applies to instances of each of the concepts named in the constraint. In this case all the concepts have the type **Role**. Since the roles **Pilot**, **Air Traffic Controller** and **Passenger** only apply to a specific real-world instance of **Flight**, this means that the same **Human** may at different times travel on a **Flight** in each of these roles. **Pilots** quite like to fly as **Passengers** when they go on holiday, for example!

Losing Our Way(point)

The second question we asked was whether the model supported different **Waypoints** with the same **Shortname**. The multiplicity between **Waypoint** and **Shortname** tells us the story. While a **Waypoint** must have one and only one **Shortname**, a particular **Shortname** can be related to more than one **Waypoint** even though they may have different **Waypoint Coordinates**.

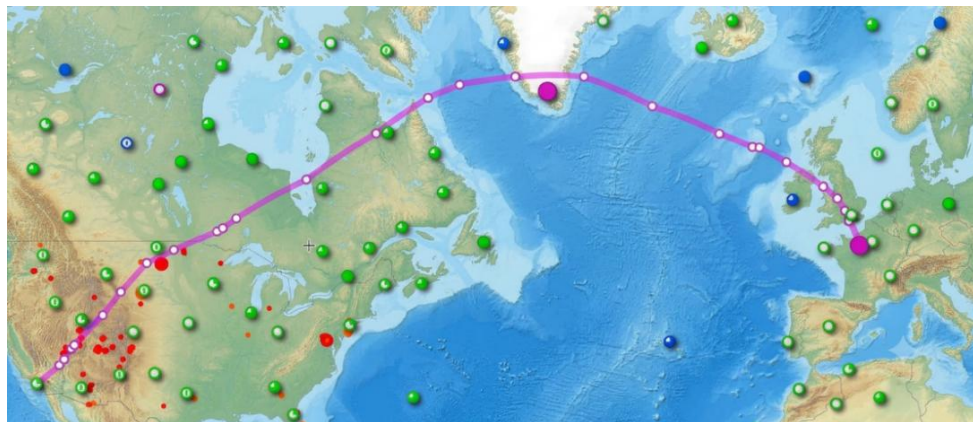


As you might guess, allowing duplicate **Waypoint Shortnames** could, and has been, problematic. In particular we will look at one such incident: the NATS Air Traffic Control (ATC) outage that occurred on the 28th Dec 2023. A duplicate **Waypoint Shortname** (**DVL** in this instance) caused a system outage that led to 700,000 passengers suffering cancellations and a cost to the industry estimated to be between £75m and £100m [7].

NATS reported that the issue occurred after 15,000,000 successful flights, so what combination of events led to the problem?

To explain the problem, we need to understand how flight planning works for international flights that might span several airspace jurisdictions, such as US, UK and French airspace. This explanation is based on one of the CAA reports of the incident [8] (CAP2993C).

The flight in question was a French Bee flight BF731, which routinely flies between Los Angeles and Paris Orly every two or three days. Normally the flight plan would be more direct across the Atlantic thus missing UK airspace entirely (this would have financial benefits) but occasionally, weather conditions dictate a more eccentric northerly route which will necessarily mean passing into UK airspace. Here is the original planned route:



As the route is unusually more northerly, it takes the flight plan, through North Dakota and hence one of the problem **Waypoints** at Devil's Lake (DVL).

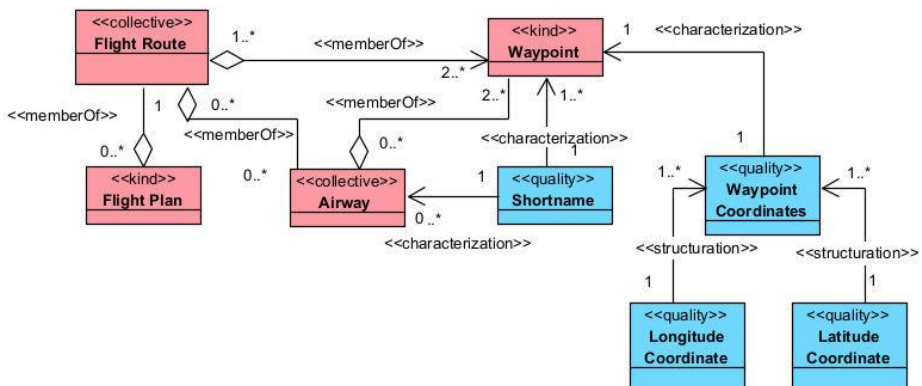
The flight plan as a collection of **Waypoints** (read from left to right) is as follows:

KLAX ORCKA5 LAS Q70 BLIPP Q842 WINEN MLF J107 OCS CZI DIK DVL VESRU
YXL ELVEL PIGLA 5870N MIBNO MAXAR 6250N 6240N 6030N 5820N SUNOT KESIX
APSOV SOSIM L15 KEPAD L151 KIDLI UN859 GWC UN859 LGL A34 BOBSA LFPO

KLAX and LFPO at the beginning and end of the sequence are the airport codes for Los Angeles and Paris Orly. Let's call this, informally, the "FB plan" for French Bee flight plan.

Note the **Waypoint Shortnames** of the form: "6030N"; these are when the aircraft travel over Atlantic regions where there are no ground-based radar or radio navigation beacons to name points after. The **Shortname** therefore takes a shortened abbreviation of the GPS position. This still fits with our model, but there is something else: look at **Waypoints** called **L15**, **L151** and **UN859**. These look like normal **Waypoint** names but they are actually **Airways** that are in themselves, a **Collective** of **Waypoints**.

These particular corridors are used so frequently along with the associated **Waypoints** that they have been summarised as a **Collective** with its own **Shortname**. Once on an **Airway**, the aircraft's flight management system will typically track through the intermediate **Waypoints** automatically. We need to adjust our model to support that possibility as follows:



So we now have a new **Class** called **Airway**, which is related to **Flight Route**, **Waypoint** and **Shortname**. The multiplicity tells us the following:

- A **Flight Route** *may* contain zero or several (note the 0..* multiplicity) **Airways** and an **Airway** *may* be included in one or more **Flight Routes**
- An **Airway** has one and only one **Shortname** and a **Shortname** may refer to one or more **Airways**. (Yes, **Airway Shortnames** are not globally unique either)
- An **Airway** has two or more **Waypoints** and a **Waypoint** may be included in one or more **Airways**

Returning to the NATS problem, you can see the **Waypoint** with **Shortname DVL** in the FB plan equating to the Devil's Lake beacon in the US.

For flights crossing European airspace, the flight plan must be submitted to EUROCONTROL for approval first. This organisation processes all flight plans requiring services from its member state Air Navigation Service Providers (ANSPs), such as NATS and their system (IFPS) converts the flight plans from an ICAO document 4444 (ICAO4444) format to another format known as ADEXP, which is a European-wide flight plan specification that includes, amongst other data, additional geographical **Waypoints** within the European region (added by EUROCONTROL) necessary to the route of a flight.

A critical **Waypoint** in the list is **APSOV**. This was in the original flight plan and denotes the entry point into UK airspace.

NATS, as part of its processing, needs to determine the exit point from UK airspace and it has some logic to do this. The logic runs through candidate exit points and discards those added by EUROCONTROL in preference to one that was in the original **Flight Plan**.

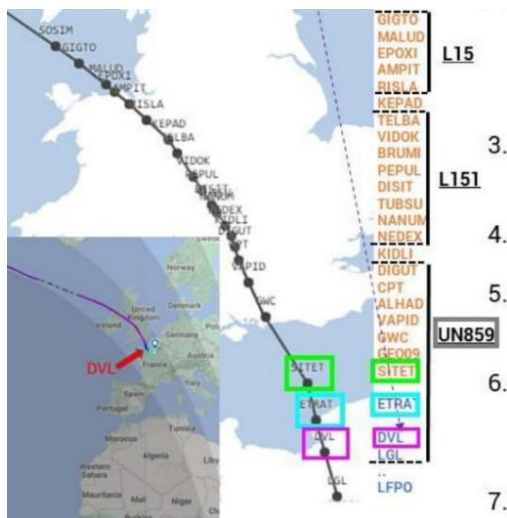
“The backup system then took over and repeated the processing resulting in the same conclusion – a critical exception, thus bringing all processing to a standstill”

As the aircraft was following **Airway** UN859, the expected exit point was **SITET**, but this was not in the original **Flight Plan** so was rejected. The next was **ETRA** and again, this was rejected as it was not in the original **Flight Plan**. The next was **DVL** and this *was* in the original **Flight Plan** but not the **DVL** the system was expecting; the system spotted that the **DVL Waypoint** was *before* the UK entry point and so couldn't possibly be a viable UK exit point so the system raised a critical exception requiring manual intervention. The backup system then took over and repeated the processing resulting in the same conclusion – a critical exception, thus bringing all processing to a standstill. Note that this is a subtle problem; it wasn't that there were two **DVL Shortnames** in the original **Flight Plan** but rather, the system processing

speculating on whether the French **DVL** was the intended exit point was tripped up by the earlier occurrence of a **Waypoint** with the same name.

From this, I'm sure you'll agree that allowing duplicate **Waypoints** exposes systems to downstream problems, even when it is known that they exist. Is **DVL** an isolated edge case of duplicate **Waypoints**? Sadly not; there are almost 4,000 duplicate **Waypoints** globally and some like **BI** have more than 10 duplicated around the world!

There are many reasons why duplicate **Waypoints** are tolerated now, which we don't have time to explore, but we'll close by simply asking how would we change our model to enforce uniqueness of **Shortnames** (across **Waypoints** and **Airways**)?

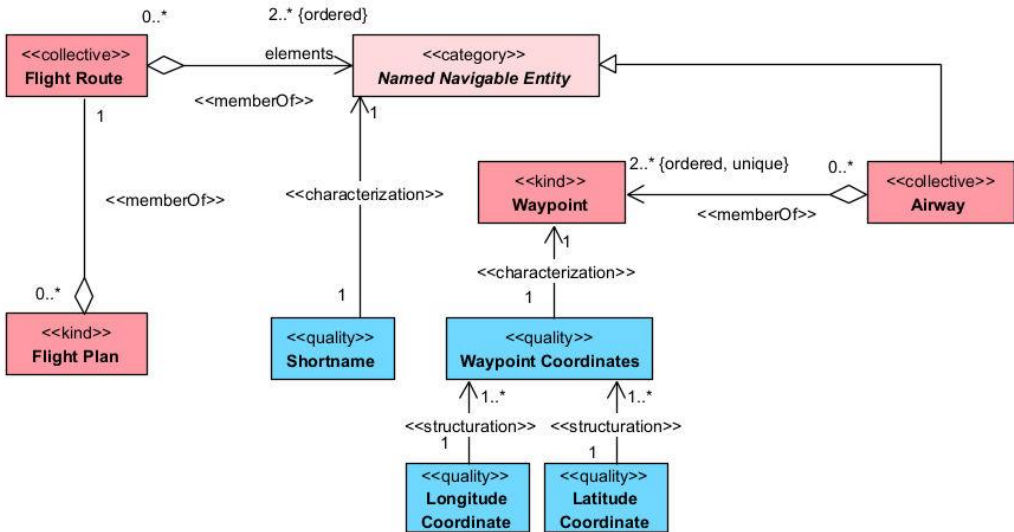


Similar problems are encountered in modelling and modern web-based systems (addressed through namespaces and Uniform Resource Identifiers). A variety of patterns have been developed and road tested to allow the same label to be used to identify multiple things without losing uniqueness.

In our case, we could try to prevent duplicates by using multiplicities, but setting the **Waypoint to Shortname** and **Airway to Shortname** relationships to 1 on both sides would not prevent a **Waypoint** and an **Airway** sharing the same **Shortname**. A neater solution is to add an abstraction of **Waypoint** and **Airway** and then add the multiplicity constraint to that higher **abstract Class**.

To do that, we will generalise the two **Classes** into an **abstract Class Named Navigable Entity**. An **abstract Class** in UML is used for things that cannot be directly instantiated, or in other words only concrete subclasses may be instantiated. Familiar examples include **Mammal**, **Fuel**, **Molecule**. In our model, as the two subtypes are different types (**Collective** and **Kind**), we introduce our final new UFO term: **Category**. **Category** can be used a little like a **Mixin** that generalise **Classes** with no common identity principle but where the underlying **Classes** are all **Rigid**; that is, **Airway** and the **Waypoint** are both a **Named Navigable Entity** for their entire lifetime. (Remember, the **Mixin Insurable Items** generalised some **Anti-Rigid Classes** such as **Pilot** – a **Phase** a **Human** may go through for part of their life).

The change is shown below again, with some diagram elements omitted for clarity.



A fairly straightforward model change that alleviates a lot of potential problems downstream!

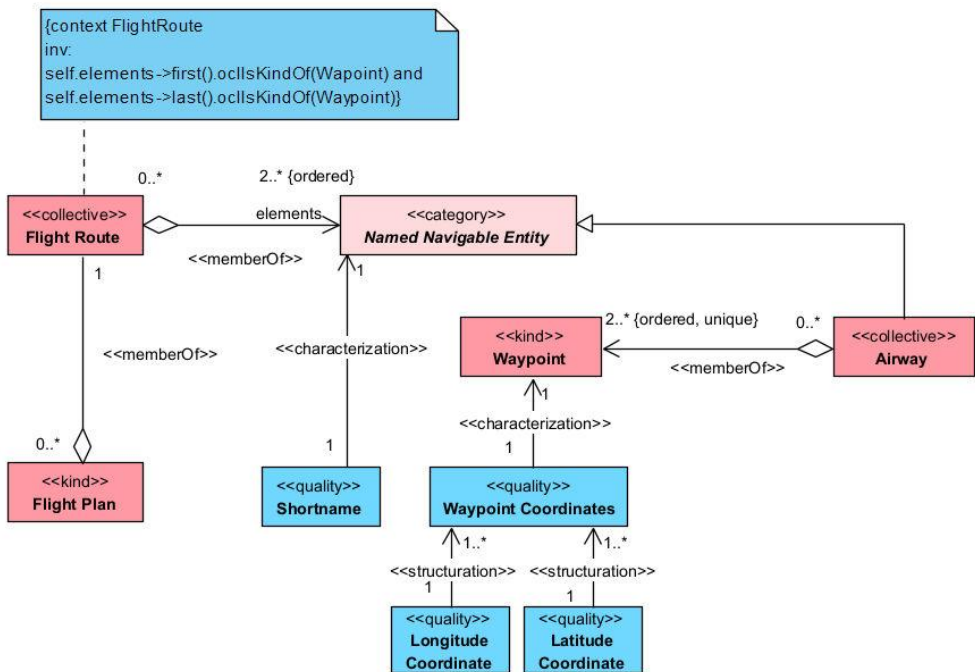
Note that we do now have a small wrinkle that we should address. In the model above, we could have a **Flight Route** formed only of **Airways**. However, a **Flight Route** must² actually start and end at an airport, aerodrome or other location designated by a **Waypoint**.

Since **Airways** never include take-off and landing point, a **Flight Route** must always start and end at a specific **Waypoint** and might (but not always) contain one or more **Airways** between those points. To resolve this, we are going to use a little OCL again on a different **Class**.

In the following diagram you will see that a new OCL constraint applies to the **Flight Route**:

```
context FlightRoute
inv:
self.elements->first().oclIsKindOf(Waypoint) and
self.elements->last().oclIsKindOf(Waypoint)
```

This is saying that in the **context** of a **Flight Route**, the first and last element must always be a single **Waypoint**. Note that, importantly, due to the way the UML language works, this constraint does not mandate that start and end **Waypoints** be different³ – if we did this, then it would prevent us modelling flights that take-off and land at the same place.



² This is a slight oversimplification which hopefully can be accepted for the purposes of this article. For example, military or emergency vehicles can potentially take off from unmapped fields.

³ UML does allow **Association Ends** to be marked **{unique}** in cases where this is desirable.

Summary

In this episode we have introduced a few more concepts such as the idea of constraints that are applied using an Object Constraint Language (OCL) and the new stereotype called **Cat-egory**, which is **Mixin**-like but used only for **Rigid Classes** with shared properties. We also saw how to expand our model when new concepts become apparent, like **Airway**. These have also helped us resolve the couple of outstanding questions we had on the model that we developed from the three previous episodes.

By exploring the NATS outage example, we've seen that despite duplicate **Waypoints** being well-known and tolerated, the devil is indeed in the detail and it is difficult to ensure that all downstream processing caters for every possible eventuality. This really underlines the importance of using ontologies (ideally at the outset): by specifying concepts in this formal way, it removes the possibility of ambiguity and the risk of misinterpretation. Initially, the model may not perfectly reflect the real-world intent, but as a means of reasoning about concepts and relationships it provides the platform for identifying shortcomings and to evolve the model so it becomes a faithful representation. With a formal ontology, it should also be easier to assess the impact of change when new concepts are introduced or existing concepts need refinement.

We really hope you have found this series of articles interesting and, of course, hope you've learned something about ontological modelling! If you would like more information on ontologies or indeed the work of the OWG then please get in touch (details are on the SCSC website: scsc.uk/owg)

References

- [1] T. R. Gruber, A Translation Approach to Portable Ontology Specifications, in Knowledge Acquisition, 5(2):199-220, Academic Press, 1993, <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>
- [2] G. Guizzardi, A. Botti Benevides, C. M. Fonseca, D. Porello, J. P. A. Almeida, T. P. Sales, Ufo: Unified foundational ontology, Applied ontology 17 (2022) 1–44, <https://doi.org/10.3233/AO-210256>
- [3] OntoUML Specification, Marek Suchánek, <https://ontouml.readthedocs.io/>, accessed Jan 2026
- [4] Unified Modelling Language, <https://www.omg.org/spec/UML>
- [5] OWG article "Modelling Tools", <https://scsc.uk/working-groups/ontology-wg-members-area/tooling>
- [6] ReOver anti-pattern, OntoUML Anti-Pattern Catalogue, <https://ontouml.readthedocs.io/en/latest/anti-patterns/ReOver/index.html>, accessed July 2026.
- [7] Air traffic shutdown sparks NATS homeworking change, <https://www.bbc.co.uk/news/articles/cev9l3dekkko> accessed July 2026.
- [8] CAA, NATS Major Incident Investigation Final Report, Flight Plan Reception Suite Automated (FPRSA-R), Sub-system Incident 28th August 2023, CAP2993C 14 Nov 2024.
- [9] Benevides, A.B., Guizzardi, G., Braga, B.F.B. and Almeida, J.P.A. (2010) 'Anti-Patterns in Ontology-Driven Conceptual Modeling', in Conceptual Modeling – ER 2010, Lecture Notes in Computer Science (LNCS), vol. 6415. Berlin, Heidelberg: Springer, pp. 317–331. https://doi.org/10.1007/978-3-642-16372-2_23

Image attributions:

leading image: Richard N Horne, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons
original planned route: Map data provided by SkyVector (www.skyvector.com) / FAA Aeronautical Information Services.

UK waypoints: NATS Major Incident Investigation Final Report (CAP2993C), UK Civil Aviation Authority, November 2024. Reproduced under Open Government Licence

The Ontology Working Group (OWG)



The OWG was set up in 2020 to help address issues with the lack of formalisation in terms and conceptual relationships used in SCSC standards and guidance documents. In particular, the similar, but at times different, interpretations of terms and concepts used between safety and security domains is an ongoing cause of confusion and misunderstandings and presents barriers to more formal integration of these disciplines.

The OWG has its origins in the Data Safety Initiative Working Group (DSIWG), which, between 2016 and 2019, explored the development of an ontology model for the concepts described in the Data Safety Guidance (DSG). However, it was realised that the work couldn't be completed without a wider ontological model for risk, and so the OWG was established and is currently working on developing a clear, systematic terminology, and unambiguous model of risk and value concepts and their relationships.

SCSC

For Everyone Working in System Safety

AI in a Safety Workflow

AI workflows fit for safety work

AI safeguards and oversight

A standards perspective of AI use

Assessment of AI-based automation

AI-based tools, methods and techniques

Evaluation and qualification of AI based tools

How assurance has to evolve in the AI era

Presenters include:

Hugues Bonnin, AUMOVIO

Ben Fulford, BMT

Sybert Stroeve, Royal Netherlands Aerospace Centre

Graham Sutherland, Consultant

29th October 2026

Hilton London Euston hotel, London, UK

www.thescsc.org

www.scsc.uk

Safety-Critical Systems Club (SCSC)

Safety-Critical Systems Club (SCSC)