NASA/TM–20250010771

Formal Structures in Systems Ontology towards Air Traffic Management Architectures

Monte Mahlum, Samantha Jarvis, Nelson Niu
NASA Langley Research Center, Hampton, Virginia

Angeline Aguinaldo, Amanda Hicks
Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland

Ian Levitt, Supervising Author
NASA Langley Research Center, Hampton, Virginia

November 2025

NASA/TM–20250010771

Formal Structures in Systems Ontology towards Air Traffic Management Architectures

Monte Mahlum, Samantha Jarvis, Nelson Niu
NASA Langley Research Center, Hampton, Virginia

Angeline Aguinaldo, Amanda Hicks
Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland

Ian Levitt, Supervising Author
NASA Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

November 2025

Available from:

NASA STI Program / Mail Stop 150
NASA Langley Research Center
Hampton, VA 23681-2199

Abstract

Driven by the need for a new aviation system architecture that can accommodate autonomy and its associated rapidly emerging technologies, NASA Langley Research Center (LaRC) and the Johns Hopkins University Applied Physics Laboratory (JHU APL) have been collaborating on a novel approach to system engineering that combines the mathematics of category theory with the philosophy of ontology. From the summer of 2024 through the summer of 2025 a group of NASA interns worked alongside NASA LaRC and JHU APL to refine the nascent approach. In this paper, the authors establish a basic framework for describing fundamental aspects of engineered systems, and for encoding that information into rigorous mathematical constructions that can be used for computations. This paper should serve as a starting point for advancing the ideas coming out of the collaboration between NASA LaRC and JHU APL, and putting them into practice.

Contents

1 Introduction

The network of controlled and uncontrolled airspace and airports in the United States operates around the clock to the tune of 45,000 flights per day, transporting more than 2.9 million passengers at a level of safety that is the gold standard for transportation. This capability is made possible only through a careful combination of personnel, processes, and technologies known as the National Airspace System (NAS) [1].

The NAS is a notoriously complex system whose behaviors are governed by an architecture of many interacting components, many of which date back to over a century. With the advent of Advanced Air Mobility (AAM) and flight operations occurring beyond visual line of sight (BVLOS) of a human operator, it is becoming broadly accepted that a critical transition of the NAS architecture is necessary to effectively accommodate new ways of operating, and to do so with even greater safety [2].

The socioeconomic system of stakeholders who are collectively and individually interested in the success of the NAS are just as complex as the NAS itself. The operators, users, regulators, researchers, developers, etc., all interact in ways that govern both the operation of the NAS and its evolution. There can be a butterfly effect in the ecosystem, wherein small misalignment between individuals and organizations lead to large unintended and deleterious system behaviors.

Shared knowledge of the system is key. The NAS of today relies on a sound foundation of formalized information like standards and regulations, with a wealth of less-structured knowledge like policies, procedures, and common practices that largely rely on the human element. The demands on NAS operations are becoming increasingly complex, and we are at the beginning of a digital transformation for aviation where machines are expected to play safety-critical roles in the NAS at scale. Abstract (but formal) models of the NAS architecture [3] can help through weaving together behaviors and capabilities, in a machine-readable way, and establishing a common vision that is understood consistently by all stakeholders.

This paper provides the ingredients for such a model, beginning with a new domain-level ontology termed the System Ontology (SO). We leverage a well-known feature of Basic Formal Ontology (BFO) called the Realization Pattern to define a schema on the realizations of a system behavior, and a mathematical construction termed a *behavior box* that serves as a formal description for how the system acts.

We will derive the SO from the BFO in Section 2. In Section 3 we will describe the requisite mathematical tools necessary for the construction of behavior boxes in Section 4. Additional information on BFO is provided in Appendix A.

2 Ontology

A powerful tool for unifying language used among experts is an *ontology*. Loosely speaking, an ontology is a framework wherein we can describe classes of entities and the relationships between them; it is scaffolding to which we can attach our knowledge and organize it. Upper-level ontologies, which can describe all types of entities, are specialized into domain-specific ontologies.

In this approach we define a specialization of an upper-level ontology, namely **Basic Formal Ontology (BFO)** [4], to a domain-specific ontology for describing systems, which we will refer to as the **Systems Ontology (SO)**.

2.1 Realization Pattern

The domain-level ontology that we will describe for systems will be a special case of the Realization Pattern from BFO, depicted in Figure 2.2. The three main types of entities in the Realization Pattern are **Process**, **Independent Continuant**, and **Realizable Entity**.

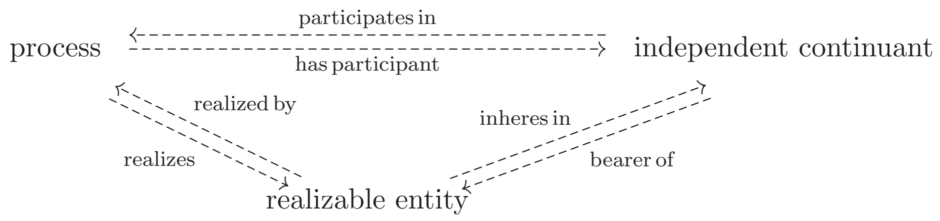

Figure 1. Realization Pattern from BFO

These three classes are disjoint, and their meanings can be more or less understood colloquially. The term *Process* hews closely to its colloquial meaning, so no further description is offered h ere. *I ndependent C ontinuants* a re u nderstood t o be objects, as opposed to processes. A *Realizable Entity* is understood to be an attribute of an object that can be actualized (or in BFO-terms, realized) through a process. More information about BFO can be found in Appendix A.

The meaning of the relations between classes can also be understood colloquially, and it is in fact the properties of these relations enumerated in Table 1 that is of particular importance. Here, *intransitive* means that the relation does not carry over across multiple links. For example, if A “has participant” B, and B “has participant” C, it does not follow that A “has participant” C. *Anti-symmetric* means that if a relation holds in one direction, it cannot hold in the opposite direction between the same two entities—for instance, if A “inheres in” B, then B cannot “inhere in” A. *Functional* means that the relation can only link an entity to one unique counterpart (e.g., a property can ”inhere in” only one bearer), whereas *non-functional* means that multiple such links are allowed (e.g., an event can be “realized by” multiple processes). Finally, an *inverse relation* captures the opposite perspective of the original relation: if A “has participant” B, then B “participates in” A.

Relation	Properties
has participant	Intransitive Anti-symmetric Non-functional Inverse relation: *participates in*
inheres in	Intransitive Anti-symmetric Functional Inverse relation: *bearer of*
realized by	Intransitive Anti-symmetric Non-functional Inverse relation: *realizes*

Table 1. Relations and their associated properties

2.2 System Ontology (SO)

In this section, we establish SO as a formal ontology for clear and consistent representation of complex systems and their behaviors. We do so by providing descriptions for **Performer**, **Disposition**, and **System Behavior** which apply to any system, and by defining how they fit within the framework of Basic Formal Ontology (BFO) and Common Core Ontology (CCO) [5].

Figure 2 below provides a simplified correspondence between the BFO and SO, showing how disposition, system behavior, and performer form a specialization of the Realization Pattern.

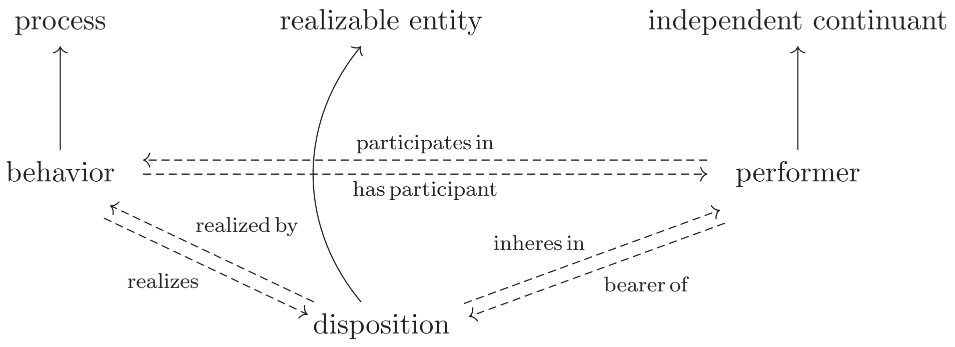

Figure 2. Systems Ontology (lower triangle) as it relates to classes in Basic Formal Ontology (top row). Again, the solid arrows depict the predicate “is a...”. For instance, ‘a disposition is a realizable entity.’

Performer. Given a system that is to be described using SO, the physical objects or beings in the system that are capable of acting or being acted upon would be considered the *performers*. In the NAS, some examples of such performers are an individual airplane, an individual pilot, or an individual Air Traffic Control (ATC).

Disposition. A *disposition* is already defined in BFO, and we use it directly in the SO to apply to systems. A disposition is understood to be an attribute, power, or potential which exists due to the physical makeup of a performer, and the performer is called its *bearer*. For instance, an aircraft has (bears) the disposition of creating powered lift, a pilot has the disposition to visually detect other aircraft, and a controller has the disposition to determine safe altitudes.

A disposition exists due to the performer's physical makeup. For example, if the airplane's engines are removed during routine maintenance then the physical makeup of the airplane would be changed and the airplane would no longer have the disposition to create powered lift.

A disposition that inheres in a performer need not always be realized. For example, the airplane has the disposition to create powered lift even when it is parked and not engaged in the process of creating powered lift.

System Behavior. We define a *system behavior* to be a process by which a disposition of some performer is realized. This is not the same as the CCO's definition of *Behavior*, however we will drop the qualifier "system" when the context is clear. For example, the behavior of flight has the aircraft as one of its participants, and the behavior of safe separation is realized partly due to the pilot's disposition to visually detect other aircraft. While the distinction between behavior and disposition is usually clear, it can sometimes be subtle. A simple way to tell them apart is to view the behavior as the process that occurs, and the disposition as an attribute of the performer that is involved.

This paper will build an initial mathematical model of systems around these three terms. Central to understanding how they tie together is the concept of *realization*. A given disposition of the system at any time may be dormant, or it may be manifest but only if the conditions for its manifestation exist. When we say that a disposition is realized by a behavior, we mean that whenever there is a particular temporal unfolding of the process where the performer that bears the disposition is actively participating and the disposition is successfully manifest.

3 Mathematics of Category Theory

Category theory is a way of looking at mathematics that focuses on the connections between ideas rather than just the details of each one. It gives us a shared language for patterns that show up across math, logic, and computer science. For a friendly introduction, Eugenia Cheng's *The Joy of Abstraction* [6] does a great job of showing both the intuition and elegance behind the subject. Another helpful resource is

Seven Sketches in Compositionality by Fong and Spivak [7], which lays out some of the foundations and offers inspiration, especially in areas like wiring diagrams.

In Section 4, we will describe a formal structure called a **behavior box** as a tool for tying together the behaviors, performers, and dispositions of a system using the SO. This will be done by applying the tools and definitions of category theory, and the relevant background is provided in this section. Specifically, we'll describe some of the basic constructions of category theory.

3.1 Preorders and categories

Before we describe what a category is, we define a mathematical structure called a preorder that will turn out to be a special case of a category.

Definition 3.1. A *preorder* $(\mathsf{P}, \leq)$ is a set P with a binary relation $\leq$ satisfying the following axioms:

- (reflexivity) for all $x \in \mathsf{P}$, we have $x \leq x$.
- (transitivity) for all $x, y, z \in \mathsf{P}$ such that $x \leq y$ and $y \leq z$, we have $x \leq z$.

A *partially ordered set* or *poset* is a preorder which also satisfies the following:

- (anti-symmetry) for all $x, y \in \mathsf{P}$, if $x \leq y$ and $y \leq x$, then $x = y$.

A *totally ordered set* is a poset which is strongly-connected:

- (strong-connectivity) for all $x, y \in \mathsf{P}$, either $x \leq y$ or $y \leq x$.

Preorders are convenient mathematical settings in which to compare two elements, although preorders that are not totally ordered may contain incomparable elements. Preorders appear in many contexts, including systems engineering. For example, when comparing the quality of two system designs, the behaviors of one may be safer while the behaviors of another may be more efficient. These designs may then be incomparable in the poset of system designs, and this is at the heart of systems engineering tradeoff analysis.

Example 3.2. The real numbers $\mathbb{R}$ with the usual ordering $\leq$ forms a totally ordered set.

Example 3.3. Let $A = \{a, b, c\}$. The power set of A, denoted $\mathcal{P}(A)$, is the set of all subsets of A:

$$\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, A\}$$

where $\emptyset$ denotes the empty set. We can order $\mathcal{P}(A)$ by inclusion, denoted by $\subseteq$ or, in the diagram below, by arrows. This is then a poset that is not totally ordered, as we cannot compare $\{a\}$ and $\{b\}$: neither $\{a\} \subseteq \{b\}$ nor $\{b\} \subseteq \{a\}$. Below is a simple figure, called a *Hasse diagram*, depicting the poset $\mathcal{P}(A)$:

$$\begin{tikzcd}
 & A & \\
\{a,b\} \arrow[ur] & \{a,c\} \arrow[u] & \{b,c\} \arrow[ul] \\
\{a\} \arrow[u] \arrow[ur] & \{b\} \arrow[ul] \arrow[ur] & \{c\} \arrow[ul] \arrow[u] \\
 & \emptyset \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

This diagram does not depict every $\subseteq$-relation in $\mathsf{P}(A)$: for example, we also have $\{a\} \subseteq A$, even though there is no arrow directly from $\{a\}$ to A. To complete this diagram, we would add an arrow between any two subsets that have a directed path between them by transitivity, as well as an arrow from every subset to itself by reflexivity. However, this diagram contains enough information to deduce all of the relations in $\mathsf{P}(A)$ via reflexivity and transitivity. For this reason, we call it a *presentation* of $\mathsf{P}(A)$.

Example 3.4. Define $\leq_{\mathbb{Z}}$ on $\mathbb{Z}$ by $n \leq_{\mathbb{Z}} m \iff |n| \leq |m|$, where $\leq$ is the usual ordering on $\mathbb{R}$ and $|\cdot|$ is the absolute value function. We then inherit reflexivity and transitivity from the order on $\mathbb{R}$, but this relation is not antisymmetric: $-1 \leq_{\mathbb{Z}} 1$ and $1 \leq_{\mathbb{Z}} -1$, but $1 \neq -1$. So $(\mathbb{Z}, \leq_{\mathbb{Z}})$ is a preorder but not a poset.

Example 3.5. There is a totally ordered set Bool with two elements $\{\text{false}, \text{true}\}$, with false $\leq$ true.

Next, we generalize the definition of a preorder to the definition of a category.

Definition 3.6. A *category* C consists of the following data:

- a collection of *objects*, also denoted by C.
- for each pair of objects $c, d \in \mathsf{C}$, a set of *morphisms* (or *arrows*) denoted by $\mathsf{C}(c,d)$ called the *hom-set* from c to d.[1] We depict a morphism $f \in \mathsf{C}(c,d)$ by $f\colon c \to d$ or $c \xrightarrow{f} d$ to emphasize that it is a morphism *from c to d*. We call c the *domain* of f and d the *codomain* of f.
- for each object $c \in \mathsf{C}$, a morphism $c \xrightarrow{\mathrm{id}_c} c$ called the *identity* on c.
- a *composition rule* which assigns to each pair of morphisms $c \xrightarrow{f} d$ and $d \xrightarrow{g} e$ a *composite* morphism $c \xrightarrow{g \circ f} e$.

These data must satisfy the following axioms:

- (identity) for all morphisms $c \xrightarrow{f} d$, we have $f \circ \mathrm{id}_c = f = \mathrm{id}_d \circ f$.

[1]Notations for the hom-set vary, and you may also see $\hom_{\mathsf{C}}(c,d)$ or $\hom(c,d)$.

- (associativity) for all composable triples h, g, f, i.e. $a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d$, we have associativity of composition: $(h \circ g) \circ f = h \circ (g \circ f)$. This allows us to unambiguously write $h \circ g \circ f$ for either composite morphism.

Example 3.7. The category of sets, denoted **Set**, has sets as objects and functions between sets as morphisms.

It turns out that all preorders are categories with a single morphism between every related pair. For instance, we can view the preorder Bool as a category as follows.

Example 3.8. The category Bool has two objects, false and true, and three morphisms: the two identity morphisms and a morphism false $\to$ true.

Example 3.9. If C is a category, then its *opposite category* C^{op} is the category with the same objects as C but with the morphisms reversed: for each morphism $c \xrightarrow{f} d$ in C, there is a corresponding arrow $d \xrightarrow{f} c$, so that the composite $c \xrightarrow{g \circ f} e$ in C of $c \xrightarrow{f} d$ and $d \xrightarrow{g} e$ corresponds to the composite $e \xrightarrow{f \circ g} c$ in C^{op} of $e \xrightarrow{g} d$ and $d \xrightarrow{f} c$.

For example, if P is a preorder with ordering $\leq$, there is an opposite preorder P^{op} with ordering $\geq$, where $p \leq q$ if and only if $q \geq p$ for all $p, q \in \mathsf{P}$.

One useful feature of category theory is that it may be used to study itself. For example, there is a category whose objects are categories and whose morphisms are called *functors*. A functor is a map between two categories that preserves their categorical structure: identities and composites. Below, C and D are categories.

Definition 3.10. A *functor* $F\colon \mathsf{C} \to \mathsf{D}$ consists of:

- an assignment to each object $c \in \mathsf{C}$ an object $F(c) \in \mathsf{D}$.
- an assignment to each morphism $a \xrightarrow{f} b$ of C a morphism $F(a) \xrightarrow{F(f)} F(b)$ of D.

These must satisfy the following axioms:

- (Identity-preserving) $F(\mathrm{id}_c) = \mathrm{id}_{F(c)}$ for all objects $c \in \mathsf{C}$.
- (Composite-preserving) $F(g \circ f) = F(g) \circ F(f)$ for all composable morphisms g, f of C.

Example 3.11. A functor $F\colon \mathsf{Bool} \to \mathsf{D}$ is precisely the data of a morphism $c \xrightarrow{f} d$ in D. More explicitly:

- F sends the object false $\in \mathsf{Bool}$ to an object $c = F(\text{false}) \in \mathsf{D}$ and the object true $\in \mathsf{Bool}$ to an object $d = F(\text{true}) \in \mathsf{D}$.
- F sends the morphism false $\to$ true of Bool a morphism $c \xrightarrow{f} d$ in D.

Example 3.12. There is a functor $\mathsf{Set} \to \mathsf{Set}$ that sends each set X to the Cartesian product $X \times X = \{(x, x') \mid x, x' \in X\}$. It then sends each function $f\colon X \to Y$ to the function $f \times f\colon X \times X \to Y \times Y$ given by:

$$(f \times f)(x, x') = (f(x), f(x')).$$

Example 3.13. If $(\mathsf{C}, \leq_{\mathsf{C}})$ and $(\mathsf{D}, \leq_{\mathsf{D}})$ are preorders viewed as categories, a functor $F\colon \mathsf{C} \to \mathsf{D}$ is an order-preserving (i.e. *monotone*) map: if $a \leq_{\mathsf{C}} b$ for $a, b \in \mathsf{C}$, then $F(a) \leq_{\mathsf{D}} F(b)$ in D.

Just as two sets may be paired up by taking their Cartesian product, two categories have a product as well.

Definition 3.14. The *product category* of C and D is a category $\mathsf{C} \times \mathsf{D}$ satisfying the following:

- its objects are ordered pairs (c, d) with $c \in \mathsf{C}$ and $d \in \mathsf{D}$.
- its morphisms $(c_1, d_1) \to (c_2, d_2)$ are ordered pairs $(c_1 \xrightarrow{f} c_2, d_1 \xrightarrow{g} d_2)$.
- composites are computed componentwise:

$$(c_2 \xrightarrow{h} c_3, d_2 \xrightarrow{k} d_3) \circ (c_1 \xrightarrow{f} c_2, d_1 \xrightarrow{g} d_2) = (c_1 \xrightarrow{h \circ f} c_3, d_1 \xrightarrow{k \circ g} d_3).$$

Example 3.15. Viewing the preorders $(\mathbb{R}, \leq)$ and Bool as categories, the product category $\mathbb{R} \times \mathsf{Bool}$ is itself a preorder. It has pairs (x, φ) for $x \in \mathbb{R}$ and $\varphi \in \mathsf{Bool}$ as objects, and $(x, \varphi) \leq (y, \psi)$ in $\mathbb{R} \times \mathsf{Bool}$ if and only if:

- $x \leq y$ and
- if φ is true, then so is ψ.

More generally, given preorders $(\mathsf{P}, \leq_{\mathsf{P}})$ and $(\mathsf{Q}, \leq_{\mathsf{Q}})$, their product is the preorder $(\mathsf{P} \times \mathsf{Q}, \leq)$ with objects (p, q) with $p \in \mathsf{P}$ and $q \in \mathsf{Q}$ satisfying $(p, q) \leq (p', q')$ if and only if $p \leq_{\mathsf{P}} p'$ and $q \leq_{\mathsf{Q}} q'$.

Finally, we provide the definition of a special kind of functor called a Bool-profunctor, which will be used in the construction of behavior boxes on the next section.

Definition 3.16. A Bool-profunctor is a functor

$$M\colon \mathsf{P}^{\mathrm{op}} \times \mathsf{R} \to \mathsf{Bool}$$

where P and R are preorders. Explicitly, if P has ordering $\leq_{\mathsf{P}}$, so that P^{op} has ordering $\geq_{\mathsf{P}}$, and R has ordering $\leq_{\mathsf{Q}}$, then M sends pairs (p, r) with $p \in \mathsf{P}$ and $r \in \mathsf{R}$ to either true or false so that:

- if $p, p' \in \mathsf{P}$ and $r, r' \in \mathsf{R}$ with $p' \leq_{\mathsf{P}} p$ and $r \leq_{\mathsf{R}} r'$, then if $M(p, r)$ is true, so is $M(p', r')$.

A Bool-profunctor $M\colon \mathsf{P}^{\mathrm{op}} \times \mathsf{R} \to \mathsf{Bool}$ is interpreted as follows. Each $r \in \mathsf{R}$ is an *input* or *resource* and each P is an *output* or *product*. Then $M(p, r)$ is true if it is *feasible* to convert r into p and false if not. Furthermore, if it is feasible to convert r into p, then it is feasible to convert a larger resource r' with $r \leq_{\mathsf{R}} r'$ in R into p. Similarly, it is also feasible to convert r into a smaller product $p' \leq_{\mathsf{P}} p$ in P.

4 Formal Structure of Architecture

In this section, we apply the mathematics from Section 3 to the SO defined in Section 2, presenting a preliminary construction that may give way to meaningful computations on system architectures. No explicit computations are presented here, instead the focus is on tools to capture the essential information about a system that could be used in such computations.

We begin by considering the definition of *system behavior* in light of it being a process having, loosely speaking, inputs and outputs. We will then organize the information associated with the occurrence of a behavior into a construction we term a *behavior box*[2]. Rather than inputs and outputs, the behavior boxes will be wired with what we will define as the *context* and the *realization*.

Using this construction, we will discuss how behaviors can be composed both in time (one behavior after another) and in detail (two behaviors combine to provide a new behavior). This will suggest areas of inquiry for developing compositional computations.

4.1 Definition of the System

Per the remark at the end of Section 2, formally defining system behaviors will hinge on the concept of the realization of its dispositions and the performers who bear them. To do this we will differentiate between classes and instances[3] of behaviors, performers, and dispositions.

We will denote classes in uppercase, and instances in lowercase. In particular a given behavior B can be treated as a class whose instances b are the various occurrences of B, and we will write $b \in B$ to indicate that b is an occurrence of B. A performer P can be treated as a class whose instances p are various physical implementation of P, and we say $p \in P$ whenever p is a specific version of P. Finally, a disposition D is a class whose instances d are the various manifestations of D and we say $d \in D$ when d is a manifestation of D.

As an example, consider a behavior $B =$ `takeoff and climb to 2,500 feet` and an occurrence $b \in B$ which is the takeoff of the plane with tail number N49 that commenced at $10:05$ June 7^{th}, 2025 and which reached 2,500 feet at $10:07$. We also have $P =$ `plane` and the particular instance $p \in P$ which is the aircraft with tail number N49. There is also a disposition $D =$ `operate at 2,500 feet`, where $d \in D$ is the particular disposition of p which was realized on June 7^{th}, 2025 at $10:07$ by b.

Note that, because D specifically depends on P, it is impossible to describe an instance d of D without also describing the $p \in P$ which bears it. We define a **description** to be a disposition-performer pair (D, P) where an instance $(d, p) \in (D, P)$ is an instance of D and an instance of P, where d is inhered in p. In order

[2]This construction is guided by similar principles and ideas as are found in the original definition of *wiring diagram* [7].

[3]It should be noted that is not necessarily aligned with the use of "instance" in the literature, and in particular needs to be resolved with the notion of universals, particulars, and types in BFO in order to complete the theory.

for (D, P) to be a valid description, it must be the case that every instance of P bears some instance of D. For example, fragility does not inhere in all vases, and so (fragility, vase) is not a valid description. We can, however, say (with high confidence) that fragility inheres in all vases that are made only of a single gram of glass and hold at least 1 liter of water.

Define a system $\mathcal{S}$ to consist of three sets, where the definitions and schema described in Section 2 apply:

1. $\mathcal{S}$(performer), of relevant performer classes,
2. $\mathcal{S}$(behavior), of relevant behavior classes, and
3. $\mathcal{S}$(disposition), of relevant disposition classes.

We impose the following requirements on the choices for $\mathcal{S}$:

- all dispositions in $\mathcal{S}$(disposition) must inhere in some performer in $\mathcal{S}$(performer), i.e., for all $D \in \mathcal{S}$(disposition), there exists $P \in \mathcal{S}$(performer) such that (D, P) is a **valid** description, and
- all behaviors in $\mathcal{S}$(behavior) must realize some disposition in $\mathcal{S}$(disposition), i.e., for all $B \in \mathcal{S}$(behavior), there exists $D \in \mathcal{S}$(disposition) and $P \in \mathcal{S}$(performer) such that (D, P) is a valid description, and there exists $b \in B$, $(d, p) \in (D, P)$ such that b realizes d.

These requirements ensure that the system is sufficiently self-contained, i.e., it does not reference any dispositions which do not inhere within the system, and it does not reference any behaviors which do not have an occurrence realizing some disposition of the system. On a practical note, we remark that the only classes in a system which must be explicitly declared are the performer classes of $\mathcal{S}$(performer), e.g., if $P =$ airplane it is not *all* airplanes that are relevant to the system and to establish this, the set corresponding to this airplane should be associated to some list of registered planes within the system database. On the contrary, all instances of $D \in \mathcal{S}$(disposition) and all occurrences of $B \in \mathcal{S}$(behavior) need not be accounted for. Instead, we build up the set of those instances and occurrences which are relevant (i.e., those which inhere in a system performer or realize a relevant disposition, respectively) via system analysis or we propose them via an architectural hypothesis.

Given this pre-conditioning of the ontological objects of a system, we may now define some associated categories.

4.2 Performer and Behavior Hierarchies

The collection of performers and behaviors are often identified by the system architect as refining hierarchies. For example, the three performers in our example architecture are `Air Traffic Controller (ATC)`, `Pilot-In-Command (PIC)`, and `Aircraft`, which are all elements of the set of all performers named the `System`, which is itself a performer. The hierarchical structure of the performers corresponds roughly to the physical decomposition of the system into subsystems, and are important for many practical reasons. We define it here as a category:

We call a given arrangement of performer hierarchy, PerHier. Evidently, if we consider the nodes in this hierarchy to be sets and the edges between nodes as set inclusion, the PerHier forms a poset. In general, we can assume to have an empty performer at the bottom of the poset, and a maximal performer, called *System*, at the top of the poset. We usually think of PerHier as being on an underlying finite set of objects representing all types of performers, called **Per**. More specifically, we have:

Definition 4.1 (PerHier)**.** PerHier is the hierarchy category of *performers.*

- The underlying set of objects is $\mathcal{S}$(performer)
- Morphisms are given by an inclusion hierarchy which form a preorder on $\mathcal{S}$(performer)

Example 4.2. An example of a given performer hierarchy, PerHier is:

```
System

        ↰ Human
                ↰ Air Traffic Controller (ATC)
                ↰ Pilot-in-Command (PIC)
        ↰ Automated
                ↰ Flight Management System (FMS)
                ↰ Airborne Collision Avoidance System (ACAS)
```

Similarly, a hierarchy of behaviors is a common and useful way to understand how a system decomposes functionally. A similar structure is defined, which we call BehHier:

Definition 4.3 (BehHier)**.** BehHier is the hierarchy category of *behaviors.*

- The underlying set of objects is $\mathcal{S}$(behavior)
- Morphisms are given by an inclusion hierarchy which form a preorder on $\mathcal{S}$(behavior)

For example, the behaviors `Detect Conflict`, `Implement Solution`, `Monitor Solution`, etc. are elements of the set called `Separation Provision`—the standard procedures exercised by ATC to prevent aircraft from conflicting.

Example 4.4. An example of a given behavior hierarchy, BehHier:

```
Separation Provision

        ↰ Detect Conflict
                ↰ Measure Distance
```

```
⮤ Formulate Solution
⮤ Implement Solution
⮤ Monitor Solution
```

An example table of dispositions associated to the performer hierarchy, PerHier, is:

Performer	**Dispositions**
System	Maintain safe separation of aircraft Optimize airspace capacity Resolve conflicts across subsystems
Human	Perceive and interpret information Communicate decisions and actions Apply judgment under uncertainty
Air Traffic Controller (ATC)	Monitor aircraft trajectories Detect potential conflicts Formulate separation solutions Issue clearances via communication
Pilot-in-Command (PIC)	Control aircraft trajectory Comply with ATC instructions Maintain awareness of weather
Automated	Process sensor data Execute programmed algorithms Generate automated advisories or commands
Flight Management System (FMS)	Compute optimal flight path Update trajectory based on constraints Interface with autopilot for trajectory
Airborne Collision Avoidance System (ACAS)	Detect intruder aircraft via transponder data Generate resolution alerts

Table 2. Performers and their associated dispositions

One can easily check that PerHier and BehHier are categories.

4.3 Context, Realization, and Behavior Boxes

For a given occurrence of a behavior $b \in B$, we now formalize what we mean by (1) the context in which b occurs and (2) the dispositions that b realizes. System behaviors unfold and exist over time, possessing temporal parts (see Definition A.3). The conditions that trigger or permit the "unfolding" or "existing" of an occurrence of a behavior we will refer to as its *context*, and the dispositions which are realized when the behavior occurs we will refer to as *realization* (outputs).

For any D, define a poset $(D, \leq)$ whose objects are the instances of d. We say that $x \leq y$ iff we somehow prefer the instance y over the instance x. This notion of goodness among instances is assumed to be assigned by the system architect, depending on the goals of the system's design. For example, consider two instances of the disposition of a banana to ripen, where one results in a perfectly ripe banana and the other results in an overripe banana. If the goal is to create a delicious banana then the first instance is preferred, but if the goal is to decompose a banana into compost then the second instance is "greater than" the first instance.

The construction of the context and realization for a behavior B of $\mathcal{S}$ proceeds as follows:

1. Define a collection $\{I_j\}_{j=1}^{n}$ (I for input) of dispositions which are relevant to the context in which B can occur. The choice of I_j should encompass all of those dispositions of the system whose realizations may impact how or if B can occur.

2. For each j, define P_j to be the performer which bears the disposition I_j. For each occurrence $b \in B$, there exists a j such that (i_j, p_j) is realized for some $i_j \in I_j$ and $p_j \in P_j$.

3. We do the same for the realization information: define a collection $\{O_j\}_{j=1}^{m}$ (O for output) of dispositions which are realized by b.

4. For each j, define Q_j to be the collection of performers which bear the disposition O_j. For each occurrence of b, there exists j such that an instance q_j of Q_j bears an instance o_j of O_j.

5. Next define the products $I := \prod_{j=1}^{n} I_j$ and $O := \prod_{j=1}^{m} O_j$ inheriting the preorder structure as described above.

We refer to the collection $\{(I_j, P_j)\}_j$ as the context, and $\{(O_j, Q_j)\}_j$ as the realization of B.

The data of when a behavior may occur, and how occurrences of that behavior instantiate the realization of dispositions, can be encapsulated in a profunctor

$$F_B : O^{\text{op}} \times I \to \mathsf{Bool},$$

which tells us when it is feasible for the behavior b to occur and have the desired effect of realizing a disposition d. More precisely,

$$F_B(o_1, o_2, ..., o_n, i_1, i_2, ..., i_m) = \textbf{true}$$

if all, and only those, descriptions

$$\{(o_j, q_j)\}_j$$

are realized whenever the descriptions

$$\{(i_j, p_j)\}_j$$

are realized and further, this process constitutes an occurrence of the behavior B.

It is worth explaining the condition encapsulated in the "and only those" parenthetical in the axiom above. This requires that for a behavior box to be valid, the behavior can be executed without realizing any dispositions other than those specified by its profunctors. The necessity of this condition is to ensure that compositionality can be verified from the data given in a behavior box.

For this construction to yield a profunctor (Definition 3.16), the preorder structures on I and O must be compatible with the feasibility axioms. We interpret the preorders as an encoding of *difficulty* in the following sense. In the case of the preorder on the context, for $i, i' \in I$, we interpret $i \leq_I i'$ to mean that it is (somehow) *easier* to execute b when i' is realized than when i is realized. In the case of the preorder on the realization, for $o, o' \in O$, we interpret $o \leq_O o'$ as it being *easier* for b to realize o' than for b to realize o.

The behavior box is depicted in the figure below.

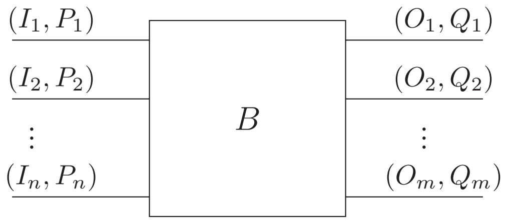

Figure 3. A depiction of a behavior box for a behavior B. The profunctor F_B tells us which instances of (O, Q) are feasible in the context of which instances of (I, P)

In Figure 4, we present a simplified example of a behavior box for the detection of a conflict by the pilot, using onboard surveillance and onboard calculations by aircraft systems.

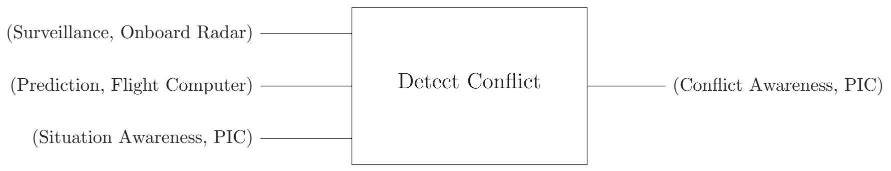
(Surveillance, Onboard Radar)
(Prediction, Flight Computer)
(Situation Awareness, PIC)
Detect Conflict
(Conflict Awareness, PIC)

Figure 4. Behavior box for `Detect Conflict`.

5 Conclusion

In the last section, we defined behavior boxes which, in principle, are able to store all information about a given behavior that is relevant to compositionality. The practicality of the approach will rely on the ability to compose and compute various profunctors that arise in the system design. There are a few areas where the theory needs further development in order to be applied.

There are multiple aspects of the SO that need more development in order for the mathematics to be well-defined. Our use of “instance” in the theory is not consistent with all conventions, and BFO has the notion of Universals, Particulars, and Instances which should be more rigorously applied. Furthermore, *role* and *quality* are important dependent continuants that need to be included alongside dispositions to form a complete theory. Finally, a sufficiently comprehensive set of specific terms (like pilot, aircraft, flight, separation provision, detect conflict) to build out enough of the ontology to be useful is an essential next step.

The posets on (instances of) dispositions is at the heart of the utility of our approach, but needs to be better understood. The notions of it being *easier* for a behavior to be executed, or *eaiser* for a behavior to realize a disposition need to be tested with more examples that test these comparisons. Crucially, more understanding is needed around what is required of these posets so that two behavior boxes can be composed when realization of one matches the context of another.

6 Acknowledgements

The authors would like to acknowledge our colleagues from NASA and JHU APL who have enriched this work with their expertise, namely—Brian Nolan, Daniel Raible, Nipa Phojanamongkolkij, Michael Castle, Walter Bender, Charles Leeper, and Eric Merrell.

References

[1] Federal Aviation Administration, *National airspace system (nas)*, `https://www.faa.gov/air_traffic/nas`, Accessed: 2025-09-17, 2025.

[2] Federal Aviation Administration, NextGen Office, "Urban air mobility (uam) concept of operations, version 2.0," U.S. Department of Transportation, Federal Aviation Administration, Tech. Report FAA-UAM-ConOps-2.0, Apr. 2023, Accessed: 2025-09-17. [Online]. Available: `https://www.faa.gov/sites/faa.gov/files/Urban%20Air%20Mobility%20%28UAM%29%20Concept%20of%20Operations%202.0_0.pdf`.

[3] Object Management Group, *Omg systems modeling language (omg sysml), version 1.6*, 2019. [Online]. Available: `https://www.omg.org/spec/SysML/1.6`.

[4] R. Arp, B. Smith, and A. D. Spear, *Building Ontologies with Basic Formal Ontology*. The MIT Press, 2015, ISBN: 9780262329583. DOI: `10.7551/mitpress/9780262527811.001.0001`.

[5] Ontology Research Group, University at Buffalo, *Common core ontologies (cco)*, `https://github.com/CommonCoreOntology/CommonCoreOntologies`, Accessed: 2025-09-17, 2018.

[6] E. Cheng, *The Joy of Abstraction: An Exploration of Math, Category Theory, and Life*. Cambridge University Press, 2022, ISBN: 978-1-108-47722-2. DOI: `10.1017/9781108769389`.

[7] B. Fong and D. I. Spivak, *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press, 2019, ISBN: 9781108711821. DOI: `10.1017/9781108668804`. [Online]. Available: `https://doi.org/10.1017/9781108668804`.

Appendix A

From Basic Formal Ontology to Systems Ontology

There are three types of entities which are central to engineered systems: performers (actors or components), system behaviors, and system dispositions. At its highest level, SO describes where these three classes fall on the BFO hierarchy and how they are interrelated. This information, along with some of the interrelations between these concepts, is depicted in Figure 2. Just as in BFO, there is a formal definition associated to each class. We now present the relevant definitions from BFO, followed by those of SO.

Ontological Definition A.1 (BFO, [4, p. 168]). A **continuant** is an entity that continues or persists through time. A continuant is said to be **independent** if it is the bearer of qualities and a participant in processes.

Remark A.2. A continuant is something that exists in full in any moment that it exists, i.e., it does not have temporal parts. The only type of independent continuant which will be relevant to us will be that of a **material entity**.

Ontological Definition A.3 (BFO, [4, p. 171]). A **process** is an entity that exists in time by occurring or happening, has temporal parts, and always depends on at least one independent continuant as participant.

Remark A.4. We will refer to each time that a process happens as a particular *occurrence* of that process. Each occurrence may have different participants.

Ontological Definition A.5 (BFO, [4, p. 169]). A **material entity** is an independent continuant that has some portion of matter as part, is spatially extended in three dimensions, and that continues to exist through some interval of time, however short.

Ontological Definition A.6 (BFO, [4, p. 171]). A **realizable entity** is a continuant that depends on an independent continuant, its bearer, for its existence and whose instances can be realized (manifested, actualized, executed) in associated processes of specific correlated types in which the bearer participates.

Remark A.7. The realization of a realizable entity is akin to the occurrence of a process. A realizable entity is "realized in bearer" during the time in which that bearer is expressing said entity. For example, a realizable entity inhering in a plane might be *flight* and this entity is realized while the plane is flying.

Ontological Definition A.8 (BFO, [4, p. 168]). A **disposition** is a realizable entity (a power, potential, or tendency) that exists because of certain features of the physical makeup of the independent continuant that is its bearer. Specifically, a disposition is a realizable entity d such that:

1. if d ceases to exist, then the bearer is physically changed,

2. d's realization occurs when and because this bearer is in some special physical circumstance, and

3. this realization occurs in virtue of the bearer's physical make-up.

Remark A.9. One can think of a disposition as an entity which, when realized, can be used to describe the process its bearer is undergoing, or the physical state therein. It is particularly the third requirement that gives us this intuitive connection to state descriptions. We emphasize this descriptional nature of dispositions with our first SO definition below.

Ontological Definition A.10 (SO)**.** A **description** is a disposition-performer pair (d, m) where the disposition d inheres in the material entity m. A description (d, m) is said to be realized whenever d is realized in m.

Remark A.11. Building on our interpretation of dispositions as a means of describing the physical state of its bearers, a description (d, m) can be thought of, when realized, as describing the state of the associated material entity m, or any other material entity in which m is contained.

Ontological Definition A.12 (SO)**.** A **behavior** is a process such that for all occurrences, a disposition of some material entity is realized, i.e., a description is realized. Each occurrence of a behavior has *actors*, those material entities which together *execute* (i.e., induce occurrence of) the behavior, and *subjects*, those for whom a disposition is realized. Together the actors and subjects can be referred to as the *performers* of the occurrence and in this occurrence they *participate* in the behavior.

The neutral language of participation, as opposed to action and subjection, is particularly useful when a behavior has all actors as subjects and vice-versa.

We now begin the process of combining the terms behavior, disposition, and performers into a definition for *system*.

As motivation, consider the observation that when defining a particular system, we often declare which performers we are considering along with some collection of relevant behaviors and dispositions. When studying a system, we often consider some compositions of our relevant behaviors and reason about those dispositions which are realized in the composite process. In engineered systems, this composite analysis is often done to study the capabilities of the system—a notion we will soon introduce into the ontology directly.

Ontological Definition A.13 (SO)**.** A **system** $\mathcal{S}$ is:

- a collection of material entities, here referred to as *performers in* $\mathcal{S}$,

- a collection of dispositions, referred to as *dispositions of* $\mathcal{S}$, each of which inheres in at least one performer in $\mathcal{S}$, and

- a collection of behaviors, referred to as *behaviors of* $\mathcal{S}$, each of which has an occurrence in which all actors are performers of $\mathcal{S}$ and at least one disposition of $\mathcal{S}$ is realized.

A description (d, p) is said to be *internal to* a system $\mathcal{S}$ if p is a performer in $\mathcal{S}$ and d is a disposition in $\mathcal{S}$.

An occurrence of a behavior is said to be *internal to* a system $\mathcal{S}$ if all actors are performers of $\mathcal{S}$ for which at least one disposition of $\mathcal{S}$ is realized.

This definition is equivalent to a more mathematical formulation presented below. We use this latter definition as it prescribes a name to each of the three sets.

Definition A.14 (System, $\mathcal{S}$)**.** A **system** $\mathcal{S}$ consists of three sets:

1. $\mathcal{S}$(performer), of relevant material entities referred to as *performers*,
2. $\mathcal{S}$(behavior), of relevant behaviors, and
3. $\mathcal{S}$(disposition), of relevant dispositions,

subject to the following requirements:

- all dispositions in $\mathcal{S}$(disposition) must inhere in some performer in $\mathcal{S}$(performer), i.e., for all $d \in \mathcal{S}$(disposition), there exists $p \in \mathcal{S}$(performer) such that d inheres in p, and
- for all behaviors in $\mathcal{S}$(behavior) there is an occurrence, whose actors are a subset of $\mathcal{S}$(performer), which realizes a disposition in $\mathcal{S}$(disposition).

These requirements ensure that the system is sufficiently self-contained, i.e., it doesn't reference any dispositions which do not inhere within the system itself and it does not reference any behaviors which are not both executable and recognizable as such by the system. Furthermore, it is only those occurrences of the behaviors which are studied in the analysis of a given system.

We allow for performers which do not participate in any relevant behaviors nor bear any relevant dispositions, however, such a system should be viewed as functionally equivalent to the same system without these extraneous performers.