A Space of

Package Delivery, Crop-dusting, Wildfire Fighting

Search & Rescue and Traffic Monitoring

Swarms and Drones

A Case Study: 12 November – 14 December, 2017

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

December 16, 2017: 08:21 am

Abstract

We speculate on a domain of *swarms* and *drones monitored* and *controlled* by a *command center* in some *geography*. Awareness of swarms is registered only in an enterprise command center. We think of these swarms of drones as an enterprise of either package deliverers, crop-dusters, insect sprayers, search & rescuers, traffic monitors, or wildfire fighters – or several of these, united in a notion of *an enterprise* possibly consisting of of "disjoint" *businesses*. We analyse & describe the properties of these phenomena as enduratns and as perdurants: parts one can observe and behaviours that one can study. We do not yet examine the problem of drone air traffic management[1]. The analysis & description of this postulated domain follows the principles, techniques and tools laid down in [30].

[2]

Experimental Engineering: This report documents an extensive, one-man, one-month exercise. The work is more engineering than science, but see the next remarks. See comments in Sects. 5.2 and 5.3 on Page 47.

Methodology Refinement: This report documents, also through its many revisions, this being **Version 12**, refinements to the method for analysing & describing manifest domains, cf. [30]. See comments in Sect. 5.3 on Page 47.

[1] https://www.nasa.gov/feature/ames/first-steps-toward-drone-traffic-management, http://www.sciencedirect.com/science/article/pii/S2046043016300260

[2] Change bars, such as you see here, indicate that the marked text is additional to or a revision of text from the immediate preceeding version.

Document History

- **Version 12** released December 16, 2017: 08:21 am.
- Eleventh 13 Dec., 2017, 17:44 CET
- Tenth 12 Dec., 2017, 16:36 CET
- Ninth 4 Dec., 2017, 6:37 am CET
- Eighth 27 Nov., 2017, 19:18 CET
- Seventh 27 Nov., 2017, 8:33 am CET
- Sixth 26 Nov., 2017, 10:47 am CET
- Fifth 21 Nov., 2017, 10:50 am CET
- Fourth 20 Nov., 2017; 15:17 CET
- Third 19 Nov., 2017.
- Second 18 Nov., 2017.
- First 14 Nov., 2017.

Warning – Version 12, December 16, 2017: 08:21 am

- Someone should play the rôle of a static checker:
  - that all used identifiers have been properly defined;
  - that types "match";
  - et cetera, et cetera !
- Some *"end are dangling"* – at least I suspect so !
- ...
- ...
- ...

Contents

0 Preamble **5**

1 An Informal Introduction **6**
1.1 Describable Entities . . . 6
1.1.1 The Endurants: Parts . . . 6
1.2 Contributions of [30] . . . 7

2 Entities, Endurants **7**
2.1 Parts, Atomic and Composite, Sorts, Abstract and Concrete Types . . . 7
2.1.1 Universe of Discourse . . . 8
2.1.2 The Enterprise . . . 8
2.1.3 From Abstract Sorts to Concrete Types . . . 9
The Auxiliary Function xtr_Ds: . . . 9
Command Center . . . 10
A Simple Narrative: . . . 10
Command Center Decomposition . . . 10
2.2 Unique Identifiers . . . 10
2.2.1 The Enterprise, the Aggregates of Drones and the Geography . . . 10
2.2.2 Unique Command Center Identifiers . . . 11
2.2.3 Unique Drone Identifiers . . . 11
Auxiliary Function: xtr_dis: . . . 11
Auxiliary Function: xtr_D: . . . 12
2.3 Mereologies . . . 12
2.3.1 Definition . . . 12
2.3.2 Origin of the Concept of Mereology as Treated Here . . . 12
2.3.3 Basic Mereology Principle . . . 12
2.3.4 Engineering versus Methodical Mereology . . . 13
2.3.5 Planner Mereology . . . 13
2.3.6 Monitor Mereology . . . 14
2.3.7 Actuator Mereology . . . 14
2.3.8 Enterprise Drone Mereology . . . 15
2.3.9 'Other' Drone Mereology . . . 16
2.3.10 Geography Mereology . . . 16
2.4 Attributes . . . 16
2.4.1 The Time Sort . . . 16
2.4.2 Positions . . . 17
A Neighbourhood Concept . . . 17
2.4.3 Flight Plans . . . 17
2.4.4 Enterprise Drone Attributes . . . 18
Constituent Types . . . 18
Attributes . . . 19
Enterprise Drone Attribute Categories: . . . 20
2.4.5 'Other' Drones Attributes . . . 20
Constituent Types . . . 20
Attributes . . . 20
2.4.6 Drone Dynamics . . . 20
2.4.7 Drone Positions . . . 21
2.4.8 Monitor Attributes . . . 21
2.4.9 Planner Attributes . . . 21
Swarms and Businesses: . . . 21
Planner Directories: . . . 21
2.4.10 Actuator Attributes . . . 23
2.4.11 Geography Attributes . . . 23
Constituent Types: . . . 23
Attributes . . . 24

3 Operations on Universe of Discourse States **24**
3.1 The Notion of a State . . . 24
3.2 Constants . . . 25
3.3 Operations . . . 25
3.3.1 A Drone Transfer . . . 25
3.3.2 Etcetera ! . . . 26

 © D.Bjørner 2017. Fredsvej 11, 2840 Holte, Denmark. December 16, 2017: 08:21

4 Perdurants **26**
4.1 System Compilation . . . 26
4.1.1 The Compile Functions . . . 27
4.1.2 Some CSP Expression Simplifications . . . 29
4.1.3 The Simplified Compilation . . . 30
4.2 An Early Narrative on Behaviours . . . 30
4.2.1 Either Endurants or Perdurants, Not Both ! . . . 30
4.2.2 Focus on Some Behaviours, Not All ! . . . 31
4.2.3 The Behaviours – a First Narrative . . . 31
4.3 Channels . . . 31
4.3.1 The Part Channels . . . 31
General Remarks: . . . 31
Part Channel Specifics . . . 32
4.3.2 Attribute Channels, General Princiles . . . 34
4.3.3 The Case Study Attribute Channels . . . 35
'Other' Drones: . . . 35
Enterprise Drones: . . . 35
Geography: . . . 35
4.4 The Atomic Behaviours . . . 35
4.4.1 Monitor Behaviour . . . 35
4.4.2 Planner Behaviour . . . 36
The Auxiliary transfer Function . . . 37
The Auxiliary flight_planning Function . . . 38
4.4.3 Actuator Behaviour . . . 39
4.4.4 'Other' Drone Behaviour . . . 40
4.4.5 Enterprise Drone Behaviour . . . 41
4.4.6 Geography Behaviour . . . 44

5 Conclusion **45**
5.1 Recent Domain Analysis & Description Case Studies . . . 46
5.2 What Am I Not Happy About ? . . . 46
5.3 What Have I Otherwise Achieved ? . . . 47
5.4 What Do I Plan to Have Done Next ? . . . 47
5.5 Acknowledgements . . . 48

6 References **48**

A Indexes **53**
A.1 Concept Definition Index . . . 53
A.2 Term Definition Index . . . 53
A.3 Formula Indexes . . . 53
A.3.1 Sorts . . . 53
A.3.2 Types . . . 53
A.3.3 Functions . . . 54
A.3.4 Channels . . . 55
A.3.5 Behaviours . . . 55
A.3.6 Constants . . . 55

B References to Swarms, Drones, UVAs, etc. **56**

0 Preamble

The background for this case study is the following: A colleague of mine, Dr Yang ShaoFa, of the *Institute of Software*, the *Chinese Academy of Sciences*, Beijing, China, "casually" mentioned, Nov. 11, 2017, "that he was preparing to think about algorithms that would advise the collective maneuvers of swarms of drones". (An example, I think, of one such algorithm, is, perhaps, that published in [46, *Declarative vs Rule-based Control for Flocking Dynamics*, Mehmood et al., April 2018].[3]) In the train, the next day, from Beijing to Xi'An, Nov. 12, I started pondering about the way I would tackle such a problem – having not paid ShaoFa proper attention to his (maybe not so casual) "mention". The present report, although far from "finished", reflects how I would approach the problem of devising and implementing such algorithms as implied by Dr Yang's 'mention'. The general problem, such as I see it, when many clever algorithms get implemented, is that they eventually appear in rather substantial (please read: "mammoth") software systems[4], systems that – for anything to work – embody millions of lines of code, distributed over hundreds of geographically widespread communications and computing systems. If the context, or as we shall here refer to it, *the domain*, is not properly understood these systems will fail – no matter how clever the algorithms are, no matter how cleverly they are implemented. And I did not want my dear colleague to witness such possibilities.

So this is then "my way" of approaching the analysis of problems and development of algorithms for even the "tiniest", seemingly well-delineated areas of computing: First *analyse &*[5] *describe the domain*. Publication [30] provides a method, its principles, techniques and tools for such analysis & description.[6] Then *develop*, as outlined in [23], in a rigorous manner, from the domain description, *a requirements prescription*. Then finally *develop the software*. By *develop* we mean: analyse & describe, validate, test, model check and formally prove properties of specifications, whether descriptions, prescriptions or code. Common abstraction and modelling principles, techniques and tools are covered in [5, 6, 7]. More good advice *et cetera* is presented in [29].

Now I present my first attempts at understanding a domain of swarms and drones. It may not be "exactly" the domain needed for Dr Yang's problem. But it may then not be too different from his needs! What I suggest by these remarks, i.e., the previous two sentences, is the following: If it is more-or-less, or almost, but not quite Dr Yang's problem domain, then change it. That is easy. Or should now be. If it is not, not at all, i.e., far from the Dr Yang's problem domain, then analyse & describe your domain – and I would strongly recommend that you do it like I have done it, using the ideas of [30]. Keep in mind the need for a subsequent requirements prescription, that is [23]. Then [30] prepares, paves the way, for a straightforward, painless requirements development.

[3] I am grateful to Dr Klaus Havelund for alerting me to the existence of this fascinating work and putting me in contact with Scott A. Smolka and Usama Mehmood.

[4] Tony Hoare is quoted as having said: *Inside very large software system there is a little program desparately trying to get out!*

[5] When we use the ampersand, '&', connecting two terms, A and B, that is $A\&B$, then we mean to designate one concept: analysis being inextricably interwoven with description, not two, as in A *and* B.

[6] Please take note, and be not surprised: I seriously think that the rôle of domain descriptions in the development of software is (1) clearly underestimated, and (2) that [30] for the first time presents a tested approach to proper software development for other than compilers. I make this seemingly bold statement on the background of the immensely successful DDC Ada compiler development project [34, 37, 47, 40]. That compiler was developed by first developing a domain description, namely the formal semantics of Ada.

 © D.Bjørner 2017. Fredsvej 11, 2840 Holte, Denmark. December 16, 2017: 08:21

1 An Informal Introduction

1.1 Describable Entities

1.1.1 The Endurants: Parts

In the universe of discourse we observe *endurants*, here in the form of parts, and *perdurants*, here in the form of behaviours.

The parts are *discrete endurants*, that is, can be seen or touched by humans, or that can be conceived as an abstraction of a discrete part.

We refer to Fig. 1.

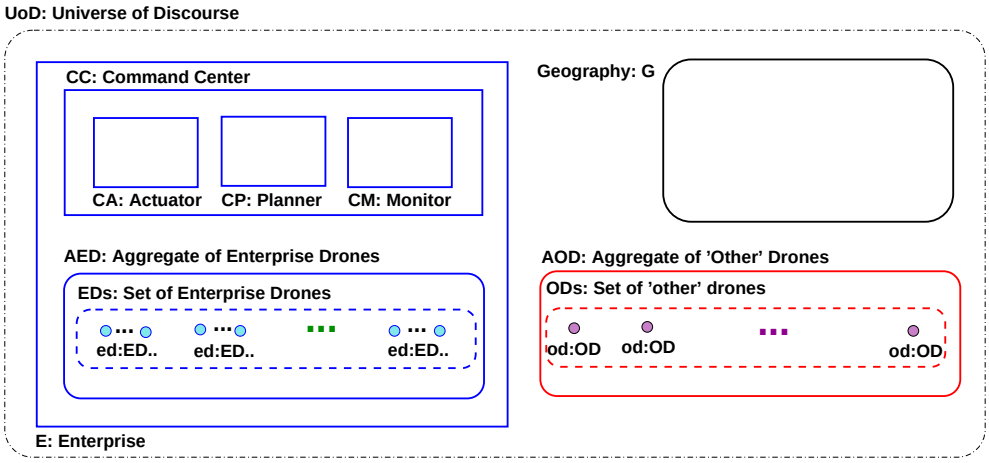

Figure 1: Universe of Discourse

There is a *universe of discourse*, uod:UoD. The universe of discourse embodies: an *enterprise*, e:E. The enterprise consists of an *aggregate of enterprise drones*, aed:AED (which consists of a set, eds:EDs, of enterprise drones). and a *command center*, cc:CC; The universe of discourse also embodies a *geography*, g:G. The universe of discourse finally embodies an *aggregate of 'other' drones*, aod:AOD (which consists of a set, ods:ODs, of these 'other' drones). A *drone* is an *unmanned aerial vehicle*.[7] We distinguish between *enterprise drones*, ed:ED, and *'other' drones*, od:OD. The `pragmatics` of the enterprise swarms is that of providing enterprise drones for one or more of the following kinds of *businesses*:[8] delivering parcels (mail, packages, etc.)[9], crop dusting[10], aerial spraying[11], wildfire fighting[12], traffic control[13], search and rescue[14], etcetera. A notion of *swarm* is introduced. A swarm is a concept. As a concept a swarm is a set of

[7] Drones are also referred to as UAVs.

[8] http://www.latimes.com/business/la-fi-drone-traffic-20170501-htmlstory.html

[9] https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011 and https://www.digitaltrends.com/cool-tech/amazon-prime-air-delivery-drones-history-progress/

[10] http://www.uavcropdustersprayers.com/, http://sprayingdrone.com/

[11] https://abjdrones.com/commercial-drone-services/industry-specific-solutions/agriculture/

[12] https://www.smithsonianmag.com/videos/category/innovation/drones-are-now-being-used-to-battle-wildfires/

[13] https://business.esa.int/sites/default/files/Presentation%20on%20UAV%20Road%20Surface%20Monitoring%20and%20Traffic%20Information_0.pdf

[14] http://sardrones.org/

drones. We associate swarms with businesses. A business has access to one or more swarms. The enterprise *command center*, cc:CC, can be seen as embodying three kinds of functions: a *monitoring* service, cm:CM, whose function it is to know the locations and dynamics of all drones, whether enterprise drones or 'other' drones; a *planning* service, cp:CP, whose function it is to plan the next moves of all that enterprise's drones; and an *actuator* service, ca:CA, whose functions it is to guide that enterprise's drones as to their next moves. The swarm concept "resides" in the command planner.

1.2 Contributions of [30]

The major contributions of [30] were these: a methodology[15] for analysing & describing manifest domains[16], where the metodology builds on an *ontological principle* of viewing the domains as consisting of *endurants* and *perdurants*. Endurants possess properties such as *unique identifiers, mereologies,* and *attributes.* Perdurants are then analysed & described as either *actions, events,* or *behaviours*. A major element in the analysis & description of behaviours is that of channels and their relation to mereology and inert, reactive and autonomous attributes. The present case study has clarified a number of issues related to channels, mereology and dynamic attributes.

• • •

The main part of this report is contained in the next three sections:endurants, Sect. 2, states, constants, and operations on states, Sect. 3 and perdurants, Sect. 4.

2 Entities, Endurants

By an *entity* we shall understand *a phenomenon, i.e., something that can be observe d, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity. We further demand that an entity can be objectively described.*

By an *endurant* we shall understand *an entity that can be observed or conceived and described as a "complete thing" at no matter which given snapshot of time. Were we to "freeze" time we would still be able to observe the entire enduranr.*

2.1 Parts, Atomic and Composite, Sorts, Abstract and Concrete Types

By a *discrete endurant* we shall understand *an endurant which is separate, individual or distinct in form or concept.*

By a *part* we shall understand *a discrete endurant which the domain engineer chooses to endow with internal qualities such as unique identification, mereology, and one or more attributes.* We shall define the concepts of unique identifier, mereology and attribute later in this report.

Atomic parts are *those which, in a given context, are deemed to not consist of meaningful, separately observable proper sub-parts.*

[15] By *methodology* we shall understand the study and knowledge abouts *methods*. By a *method* we shall understand a set of *principles* for `selecting` and `applying` a number of *techniques*, using *tools*, to – in this case – analyse & describe a domain.

[16] A manifest domain is a human- and artifact-assisted arrangement of endurant, that is spatially "stable", and perdurant, that is temporally "fleeting" entities. Endurant entities are either parts or components or materials. Perdurant entities are either actions or events or behaviours.

Sub-parts are parts.

Composite parts are *those which, in a given context, are deemed to indeed consist of meaningful, separately observable proper sub-parts.*

By a *sort* we shall understand *an abstract type.*

By a *type* we shall here understand *a set of values "of the same kind"* – where we do not further define what we mean by *the same kind".*

By an *abstract type* we shall understand *a type about whose values we make no assumption* [as to their atomicity or composition.

By a *concrete type* we shall understand *a type about whose values we are making certain assumptions as to their atomicity or composition, and, if composed then how and from which other types they are composed.*

2.1.1 Universe of Discourse

By a *universe of discourse* we shall understand *that which we can talk about, refer to and whose entities we can name.* Included in that universe is the *geography.* By *geography* we shall understand a section of the globe, an area of land, its geodecy, its meteorology, etc.

1 In the **U**niverse **o**f **D**iscourse we can observe the following parts:

   (a) an atomic **G**eography,

   (b) a composite **E**nterprise,

   (c) and an aggregate of '**O**ther'[17] **D**rones.

type
1 UoD, G, E, AOD
value
1(a) obs_G: UoD → G
1(b) obs_E: UoD → E
1(c) obs_AOD: UoD → AOD

2.1.2 The Enterprise

2 From an enterprise one can observe:

   (a) a(n enterprise) command center. and

   (b) an aggregate of enterprise drones.

type
2(a) CC
2(a) AED
value
2(a) obs_CC: E → CC
2(b) obs_AED: E → AED

[17] We apologize for our using the term 'other' drones. These 'other' drones are not necessarilt adversary or enemy drones. They are just there – coexisting with the enterprise drones.

2.1.3 From Abstract Sorts to Concrete Types

3 From an *aggregate of enterprise drones*, AED, we can observe a possibly empty set of drones, EDs

4 From an *aggregate of 'other' drones*, AOD, we can observe a possibly empty set, ODs, of *'other' drones*.

type
3 ED
3 EDs = ED-**set**
4 OD
4 ODs = OD-**set**
value
3 obs_EDs: AED → EDs
4 obs_ODs: AOD → ODs

Drones, whether 'other' or 'enterprise', are considered atomic.

The Auxiliary Function xtr_Ds: We define an auxiliary function, xtr_Ds.

5 From the universe of discourse we can extract all its drones;

6 similarly from its enterprise;

7 similarly from the aggregate of enterprise drones; and

8 from an aggregate of 'other' drones.

5 xtr_Ds: UoD → (ED|OD)-**set**
5 xtr_Ds(uod) ≡
5 ∪{xtr_Ds(obs_AED(obs_E(uod)))} ∪ xtr_Ds(obs_AOD(uod))
6 xtr_Ds: E → ED-**set**
6 xtr_Ds(e) ≡ xtr_Ds(obs_AED(e))
7 xtr_Ds: AED → ED-**set**
7 xtr_Ds(aed) ≡ obs_EDs(obs_EDs(aed))
8 xtr_Ds: AOD → OD-**set**
8 xtr_Ds(aod) ≡ obs_ODs(aod)

9 In the universe of discourse a drone cannot be both among the enterprise drones and among the 'other' drones.

axiom
9 ∀ uod:UoD,e:E,aed:ES,aod:AOD •
9 e=obs_E(uod)∧aed=obs_AED(e)∧aod:obs_AOD(uod)
9 ⇒ xtr_Ds(aed) ∩ xtr_Ds(aod) ={}

The functions are partial as the supplied swarm identifier may not be one of the universe of discourse, etc.

 © D.Bjørner 2017. Fredsvej 11, 2840 Holte, Denmark. December 16, 2017: 08:21

Command Center

A Simple Narrative: Figure 1 on Page 6 shows a graphic rendition of a space of interest. The command center, CC, a composite part, is shown to include three atomic parts: An atomic part, the monitor, CM. It monitors the location and dynamics of all drones. An atomic part, the planner, CP. It plans the next, "friendly", drone movements. The command center also has yet an atomic part, the actuator, CA. It informs "friendly" drones of their next movements. The planner is where "resides" the notion of a enterprise consisting of one or more businesses, where each business has access to zero, one or more swarms, where a swarm is a set of enterprise drone identifiers.
The purpose of the control center is to monitor the whereabouts and dynamics of all drones (done by CM); to plan possible next actions by enterprise drones (done by CP); and to instruct enterprise drones of possible next actions (done by CA).

Command Center Decomposition From the composite command center we can observe

10 the center monitor, CM;

11 the center planner, CP; and

12 the center actuator, CA .

type
10 CM
11 CP
12 CA

value
10 obs_CM: CC → CM
11 obs_CP: CC → CP
12 obs_CA: CC → CA

2.2 Unique Identifiers

Parts are distinguishable through their unique identifiers. A *unique identifier* is a further undefined quantity which we associate with parts such that no two parts of a universe of discourse are identical.

2.2.1 The Enterprise, the Aggregates of Drones and the Geography

13 Although we may not need it for subsequent descriptions we do, for completeness of description, introduce unique identifiers for parts and sub-parts of the universe of discourse:

(a) Geographies, g:G, have unique identification.
(b) Enterprises, e:E, have unique identification.
(c) Aggregates of enterprise drones, aed:AED, have unique identification.
(d) Aggregates of 'other' drones, aod:AOD, have unique identification.
(e) Command centers, cc:CC, have unique identification.

type
13 GI, EI, AEDI, AODI, CCI
value

13(a) uid_G: G → GI
13(b) uid_E: E → EI
13(c) uid_AED: AED → AEDI
13(d) uid_OD: AOD → AODI
13(e) uid_CC: CC → CCI

2.2.2 Unique Command Center Identifiers

14 The monitor has a unique identifier.

15 The planner has a unique identifier.

16 The actuator has a unique identifier.

type
14 CMI
15 CPI
16 CAI

value
14 uid_CM: CM → CMI
15 uid_CP: CP → CPI
16 uid_CA: CA → CAI

2.2.3 Unique Drone Identifiers

17 Drones have unique identifiers.

(a) whether enterprise or
(b) 'other' drones

type
17 DI = EDI | ODI
value
17(a) uid_ED: ED → EDI
17(b) uid_OD: OD → ODI

Auxiliary Function: xtr_dis:

18 From the aggregate of enterprise drones;

19 From the aggregate of 'other' drones;

20 and from the two parts of a universe of discourse: the enterprise and the 'other' drones.

value
18 xtr_dis: AED → DI-**set**
18 xtr_dis(aed) ≡ {uid_ED(ed)|ed:ED•ed ∈ obs_EDs(aed)}
19 xtr_dis: AOD → DI-**set**
19 xtr_dis(aod) ≡ {uid_D(od)|od:OD•od ∈ obs_ODs(aod)}
20 xtr_dis: UoD → DI-**set**
20 xtr_dis(uod) ≡ xtr_dis(obs_AED(uod)) ∪ xtr_dis(obs_AOD(uod))

 © D.Bjørner 2017. Fredsvej 11, 2840 Holte, Denmark. December 16, 2017: 08:21

Auxiliary Function: xtr_D:

21 From the universe of discourse, given a drone identifier of that space, we can extract the identified drone;

22 similarly from the enterprise;

23 its aggregate of enterprise drones; and

24 and from its aggregate of 'other' drones;

21 xtr_D: UoD → DI $\xrightarrow{\sim}$ D
21 xtr_D(uod)(di) ≡ **let** d:D • d ∈ xtr_Ds(uod)∧uid_D(d)=di **in** d **end**
21 **pre**: di ∈ xtr_dis(soi)
22 xtr_D: E → DI $\xrightarrow{\sim}$ D
22 xtr_D(e)(di) ≡ **let** d:D • d ∈ xtr_Ds(obs_ES(e))∧uid_D(d)=di **in** d **end**
22 **pre**: di ∈ xtr_dis(e)
23 xtr_D: AED → DI $\xrightarrow{\sim}$ D
23 xtr_D(aed)(di) ≡ ≡ **let** d:D • d ∈ xtr_Ds(aed)∧uid_D(d)=di **in** d **end**
23 **pre**: di ∈ xtr_dis(es)
24 xtr_D: AOD → DI $\xrightarrow{\sim}$ D
24 xtr_D(aod)(di) ≡ **let** d:D • d ∈ xtr_Ds(aod)∧uid_D(d)=di **in** d **end**
24 **pre**: di ∈ xtr_dis(ds)

2.3 Mereologies

2.3.1 Definition

Mereology is the study and knowledge of parts and their relations (to other parts and to the "whole") [36].

2.3.2 Origin of the Concept of Mereology as Treated Here

We shall [thus] deploy the concept of mereology as advanced by the Polish mathematician, logician and philosopher Stanisław Léschniewski. Douglas T. ("Doug") Ross[18] also contributed along the lines of our approach [54] – hence [27] is dedicated to Doug.

2.3.3 Basic Mereology Principle

The basic principle in modelling the mereology of a any universe of discourse is as follows: Let p' be a part with unique identifier p'_{id}. Let p be a sub-part of p' with unique identifier p_{id}. Let the immediate sub-parts of p be $p_1, p_2, \ldots, p_n$ with unique identifiers $p_{1_{id}}, p_{2_{id}}, \ldots, p_{n_{id}}$. That p has mereology $(p'_{id}, \{p_{1_{id}}, p_{2_{id}}, \ldots, p_{n_{id}}\})$. The parts p_j, for $1 \leq j \leq n$ for $n \geq 2$, if atomic, have mereologies $(p_{id}, \{p_{1_{id}}, p_{2_{id}}, \ldots, p_{j-1_{id}}, p_{j+1_{id}}, \ldots, p_{n_{id}}\})$ – where we refer to the second term in that pair by m; and if composite, have mereologies $(p_{id}, (m, m'))$, where the m' term is the set of unique identifiers of the sub-parts of p_j.

[18] Doug Ross is the originator of the term CAD for *computer aided design*, of APT for *Automatically Programmed Tools*, a language to drive numerically controlled manufacturing, and also SADT for *Structure Analysis and Design Techniques*

2.3.4 Engineering versus Methodical Mereology

We shall restrict ourselves to an engineering treatment of the mereology of our universe of discourse. That is in contrast to a strict, methodical treatment. In a methodical description of the mereologies of the various parts of the universe of discourse one assigns a mereology to every part: to the enterprise, the aggregate of 'other' drones and the geography; to the command center of the enterprise and its aggregate of drones; to the monitor, the planner and the actuator of the command center; to the drones of the aggregate of enterprise drones, and to the drones of the aggregate of 'other' drones. We shall "shortcut" most of these mereologies. The reason is this: The *pragmatics* of our attempt to model *drones*, is rooted in our interest in the interactions between the command center's monitor and actuator and the enterprise and 'other' drones. For "completeness" we also include interactions between the geography's meteorology and the above command center and drones. The mereologies of the enterprise, E, the enterprise aggregate of drones AED, and the set of (enterprise) drones, EDs, do not involve drone identifiers. The only "thing" that the monitor and actuator are interested in are the drone identifiers. So we shall thus model the mereologies of our universe of discourse by omitting mereologies for the enterprise, the aggregates of drones, the sets of these aggregates, and the geography, and only describe the mereologies of the monitor, planner and actuator, the enterprise drones and the 'other' drones.

2.3.5 Planner Mereology

25 The planner mereology reflects the center planners awareness[19] of the monitor, the actuator,, and the geography of the universe of discourse.

26 The plannner mereology further reflects that a *eureka*[20] is provided by, or from, an outside source reflected in the autonomous attribute CmdI. The value of this attribute changes at its own volition and ranges over commands that directs the planner to perform either of a number of operations.

Eureka examples are: calculate and effect a new flight plan for one or more designated swarms of a designated business; effect the transfer of an enterprise drone from a designated swarm of a business to another, distinctly designated swarm of the same business; etcetera.

```
type
25  CPM = (CAI × CMI × GI) × Eureka
26  Eureka == mkNewFP(BI×SI-set×Plan)
26          | mkChgDB(fsi:SI×tsi:SI×di×DI)
26          | ...
value
25  mereo_CP: CP → CPM
26  Plan = ...
```

[19] That "awareness" includes, amongst others, the planner obtaining information from the monitor of the whereabouts of all drones and providing the actuator with directives for the enterprise drones — all in the context of the *land* and "its" *meteorology*.

[20] "Eureka" comes from the Ancient Greek word $\epsilon\mu\rho\eta\kappa\alpha$ `heúrēka`, meaning "I have found (it)", which is the first person singular perfect indicative active of the verb $\epsilon\upsilon\rho\eta\kappa\omega$ `heuriskō` "I find".[1] It is closely related to heuristic, which refers to experience-based techniques for problem solving, learning, and discovery.

We omit expressing a suitable axiom concerning center planner mereologies. Our behavioural analysis & description of monitoring & control of operations on the space of drones will show that command center mereologies may change.

2.3.6 Monitor Mereology

The monitor's mereology reflects its awareness of the drones whose position and dynamics it is expected to monitor.

27 The mereology of the center monitor is a pair: the set of unique identifiers of the drones of the universe of discourse, and the unique identifier of the center planner.

type
27 CMM = DI-**set** × CPI
value
27 mereo_CM: CM → CMM

28 For the universe of discourse it is the case that

(a) the drone identifiers of the mereology of a monitor must be exactly those of the drones of the universe of discourse, and

(b) the planner identifier of the mereology of a monitor must be exactly that of the planner of the universe of discourse.

axiom
28 ∀ uod:UoD,e:E,cc:CC,cp:CP,cm:CM,g:G •
28 e=obs_E(uod)∧cc=obs_CC(e)∧cp=obs_CP(cc)∧cm=obs_CM(cc) ⇒
28 **let** (dis,cpi) = mereo_CM(cm) **in**
28(a) dis = xtr_dis(uod)
28(b) ∧ cpi = uid_CP(cp) **end**

2.3.7 Actuator Mereology

The center actuator's mereology reflects its awareness of the enterprise drones whose position and dynamics it is expected to control.

29 The mereology of the center actuator is a pair: the set of unique identifiers of the business drones of the universe of discourse, and the unique identifier of the center planner.

type
29 CAM = EDI-**set** × CPI
value
29 mereo_CA: CA → CAM

30 For all universes of discourse

(a) the drone identifiers of the mereology of a center actuator must be exactly those of the enterprise drones of the space of interest (of the monitor), and

(b) the center planner identifier of the mereology of a center actuator must be exactly that of the center planner of the command center of the space of interest (of the monitor)

axiom
30 ∀ uod:UoD,e:E,cc:CC,cp:CP,ca:CA •
30 e=obs_E(uod)∧cc=obs_CC(e)∧cp=obs_CP(cc)∧ca=obs_CA(cc) ⇒
30 **let** (dis,cpi) = mereo_CA(ca) **in**
30(a) dis = tr_dis(e)
30(b) ∧ cpi = uid_CP(cp) **end**

2.3.8 Enterprise Drone Mereology

31 The mereology of an enterprise drone is the triple of the command center monitor, the command center actuator[21], and the geography.

type
31 EDM = CMI × CAI × GI
value
31 mereo_ED: ED → EDM

32 For all universes of discourse the enterprise drone mereology satisfies:

(a) the unique identifier of the first element of the drone mereology is that of the enterprise's command monitor,

(b) the unique identifier of the second element of the drone mereology is that of the enterprise's command actuator, and

(c) the unique identifier of the third element of the drone mereology is that of the universe of discourse's geography.

axiom
32 ∀ uod:UoD,e:E,cm:CM,ca:CA,ed:ED,g:G •
32 e=obs_E(uod)∧cm=obs_CM(obs_CC(e))∧ca=obs_CA(obs_CC(e))
32 ∧ ed ∈ xtr_Ds(e)∧g=obs_G(uod) ⇒
32 **let** (cmi,cai,gi) = mereo_D(ed) **in**
32(a) cmi = uid_CMM(ccm)
32(b) ∧ cai = uid_CAI(cai)
32(c) ∧ gi = uid_G(g) **end**

[21] The command center monitor and the command center actuator and their unique identifiers will be defined in Items 10, 12 on Page 10, 14 and 16 on Page 11.

2.3.9 'Other' Drone Mereology

33 The mereology of an 'other' drone is a pair: the unique identifier of the monitor and the unique identifier of the geography.

type
33 ODM = CMI × GI
value
33 mereo_OD: OD → ODM

We leave it to the reader to formulate a suitable axiom, cf. axiom 32 on the preceding page.

2.3.10 Geography Mereology

34 The geography mereology is a pair[22] of the unique of the unique identifiers of the planner and the set of all drones.

type
34 GM = CPI × CMI × DI-**set**
value
34 mereo_G: G → GM

We leave it to the reader to formulate a suitable axiom, cf. axiom 32 on the previous page.

2.4 Attributes

We analyse & describe attributes for the following parts: *enterprise drones* and *'other' drones*, *monitor, planner* and *actuator*, and the *geography*. The attributes, that we shall arrive at, are usually concrete in the sense that they comprise values of, as we shall call them, *constituent* types. We shall therefore first analyse & describe these constituent types. Then we introduce the part attributes as expressed in terms of the constituent types. But first we introduce three notions core notions: time, Sect. 2.4.1, positions, Sect. 2.4.2, and flight plans, Sect. 2.4.3.

2.4.1 The Time Sort

35 Let the special sort identifier $\mathbb{T}$ denote times

36 and the special sort identifier $\mathbb{TI}$ denote time intervals.

37 Let identifier time designate a "magic" function whose invocations yield times.

type
35 $\mathbb{T}$
35 $\mathbb{TI}$
value
35 time: **Unit** → $\mathbb{T}$

[22] 30.11.2017: I think!

38 Two times can not be added, multiplied or divided, but subtracting one time from another yields a time interval.

39 Two times can be compared: smaller than, smaller than or equal, equal, not equal, etc.

40 Two time intervals can be compared: smaller than, smaller than or equal, equal, not equal, etc.

41 A time interval can be multiplied by a real number.

Etcetera.

value
38 $\ominus$: $\mathbb{T} \times \mathbb{T} \to \mathbb{TI}$
39 $<,\leq,=,\neq,\geq,>$: $\mathbb{T} \times \mathbb{T} \to$ **Bool**
40 $<,\leq,=,\neq,\geq,>$: $\mathbb{TI} \times \mathbb{TI} \to$ **Bool**
41 $\otimes$: $\mathbb{TI} \times$ **Real** $\to \mathbb{TI}$

2.4.2 Positions

Positions (of drones) play a pivotal rôle.

42 Each *position* being designated by

43 *longitude, latitude* and *altitude*.

type
43 LO, LA, AL
42 P = LO × LA × AL

A Neighbourhood Concept

44 Two positions are said to be *neighbours* if the *distance* between them is small enough for a drone to fly from one to the other in one to three minutes' time – for drones flying at a speed below `Mach 1.`

value
44 neighbours: P × P → **Bool**

We leave the neighbourhood proposition further undefined.

2.4.3 Flight Plans

A crucial notion of our universe of discourse is that of flight plans.

45 A *flight plan element* is a pair of a time and a position.

46 A *flight plan* is a sequence of flight plan elements.

type
45 FPE = $\mathbb{T}$ × P
46 FP = FLE*

47 such that adjacent entries in flight plans

(a) record increasing times and

(b) neighbouring positions.

axiom
47 ∀ fp:FP,i:**Nat** • {i,i+1}⊆**inds**fp ⇒
47 **let** (t,p)=fp[i], (t′,p′)=fp[i+1] **in**
47(a) t ≤ t′
47(b) ∧ neighbours(p,p′)
47 **end**

2.4.4 Enterprise Drone Attributes

Constituent Types

48 Enterprise drones have *positions* expressed, for example, in terms of *longitude, latitude* and *altitude*. [23]

49 Enterprise drones have *velocity* which is a vector of *speed* and three-dimensional, i.e., spatial, *direction*.

50 Enterprise drones have *acceleration* which is a vector of *increase/decrease of speed per time unit* and *direction*.

51 Enterprise drones have orientation which is expressed in terms of three quantities: *yaw, pitch* and *roll*.[24]

We leave *speed, direction* and *increase/decrease per time unit* unspecified.

type
48 POS = P
49 VEL = SPEED × DIRECTION

[23] *Longitude* is a geographic coordinate that specifies the east-west position of a point on the Earth's surface. It is an angular measurement, usually expressed in degrees and denoted by the Greek letter lambda. Meridians (lines running from the North Pole to the South Pole) connect points with the same longitude. *Latitude* is a geographic coordinate that specifies the northsouth position of a point on the Earth's surface. Latitude is an angle (defined below) which ranges from 0^o at the Equator to 90^o (North or South) at the poles. Lines of constant latitude, or parallels, run eastwest as circles parallel to the equator. *Altitude* or height (sometimes known as depth) is defined based on the context in which it is used (aviation, geometry, geographical survey, sport, and many more). As a general definition, altitude is a distance measurement, usually in the vertical or "up" direction, between a reference datum and a point or object. The reference datum also often varies according to the context.

[24] *Yaw, pitch* and *roll* are seen as symmetry axes of a drone: normal axis, lateral (or transverse) axis and longitudinal (or roll) axis. See Fig. 2 on the facing page.

50 ACC = IncrDecrSPEEDperTimeUnit × DIRECTION
51 ORI = YAW × PITCH × ROLL
49 SPEED = ...
49 DIRECTION = ...
50 IncrDecrSPEEDperTimeUnit = ...

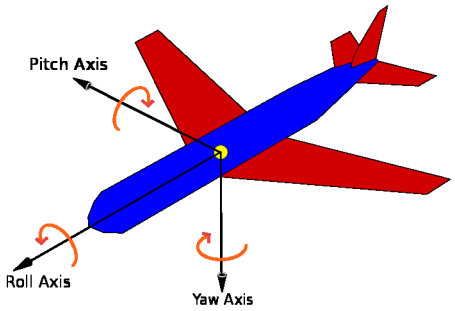

Figure 2: Aircraft Orientation

Attributes

52 One of the enterprise properties is that of its *dynamics* which is seen as a quadruple of *velocity, acceleration, orientation* and *position*. It is recorded as a reactive attribute.

53 Enterprise drones follow a flight course, as prescribed in and recorded as a programmable attribute, referred to a the *future flight plan*, FFP.

54 Enterprise drones have followed a course recorded, also a programmable attribute, as a *past flight plan list*, PFPL.

55 Finally enterprise drones "remember", in the form of a programmable attribute, the *geography* (i.e., the *area*, the *land* and the *weather*) it is flying over and in!

type
55 ImG = A×L×W
52 DYN = s_vel:VEL × s_acc:ACC × s_ori:ORI × s_pos:POS
53 FPL = FP
54 PFPL = FP*
value
52 attr_DYN: ED → DYN
53 attr_FPL: ED → FPL
54 attr_PFPL: ED → PFPL
55 attr_ImG: ED → ImG

Enterprise, as well as 'other' drone, positions must fall within the *Euclidian Point Space* of the geography of the universe of discourse. We leave that as an axiom to be defined – or we could decide that if a drone leaves that space then it is lost, and if drones suddenly "appear, out of the blue", then they are either "brand new", or "reappear".

Enterprise Drone Attribute Categories: The position, velocity, acceleration, position and past position list attributes belong to the **reactive** category. The future position list attribute belong to the **programmable** category. Drones have a "zillion" more attributes – which may be introduced in due course.

2.4.5 'Other' Drones Attributes

Constituent Types The constituent types of 'other' drones are similar to those of some of the enterprise drones.

Attributes

56 'Other' drones have *dynamics*, dyn:DYN.

57 'Other' drones "remember", in the form of a programmable attribute, the *immediate geography*, ImG (i.e., the *area*, the *land* and the *weather*) it is flying over and in!

type
57 A, L, W
57 ImG = A×L×W
value
56 attr_DYN: OD → DYN
57 attr_ImG: OD → ImG

2.4.6 Drone Dynamics

58 By a timed drone dynamics, TiDYN, we understand a quadruplet of *time, position, dynamics* and *immediate geography*.

59 By a *current drone dynamics* we shall understand a drone identifier-indexed set of timed drone dynamics.

60 By a *record of [traces of] timed drone dynamics* we shall understand a drone identifier-indexed set of sequences of timed drone dynamics.

type
58 TiDYN = $\mathbb{T}$ × POS × DYN × ImG
59 CuDD = (EDI $\overrightarrow{m}$ TiDYN) ∪ (ODI $\overrightarrow{m}$ TiDYN)
60 RoDD = (EDI $\overrightarrow{m}$ TiDYN*) ∪ (ODI $\overrightarrow{m}$ TiDYN*)

We shall use the notion of *current drone dynamics* as the means whereby the *monitor* ascertains (obtains, by interacting with drones) the dynamics of drones, and the notion of a *record of [traces of] drone dynamics* in the *monitor*.

2.4.7 Drone Positions

61 For all drones whether enterprise or 'other', their positions must lie within the geography of their universe of discourse.

axiom
61 ∀ uod:UoD,e:E,g:G,d:(ED|OD) •
61 e = obs_E(uod) ∧ g = obs_G(uod) ∧ d ∈ xtr_Ds(uod) ⇒
61 **let** eps = attr_EPS(g), (_,_,p) = attr_DYN(d) **in** p ∈ eps **end**

2.4.8 Monitor Attributes

The *monitor* "sits between" the *drones* whose dynamics it monitors and the *planner* which it provides with records of drone dymamics. Therefore we introduce the following.

62 The monitor has just one, a programmable attribute: a trace of the most recent and all past time-stamped recordings of the dynamics of all drones, that is, an element rodd:RoDD, cf. Item 60 on the preceding page .

type
62 MRoDD = RoDD
value
62 attr_MRoDD: CM → MRoDD

The monitor "obtains" current drone dynamics, cudd:CuDD (cf. Item 59 on the facing page) from the *drones* and offers records of [traces of] drone dynamics,(cf. Item 60 on the preceding page) rodd:RoDD, to the *planner*.

2.4.9 Planner Attributes

Swarms and Businesses: The *planner* is where all decisions are made with respect to where enterprise drones should be flying; which enterprise drones fly together, which no longer – (with this notion of "flying together" leading us to the concept of *swarms*); which swarms of enterprise drones do which kinds of work – (with this notion of work specialisation leading us to the concept of businesses.)

63 The is a notion of a *business identifier*, BI.

type
63 BI

Planner Directories: Planners have three directories. These are attributes, BDIR (businesses), SDIR (swarms) and DDIR (drones).

64 BDIR records which swarms are resources of which businesses;

65 SDIR records which drones "belong" to which swarms.

66 DDIR "keeps track" of past and present enterprise drone positions, as per enterprise drone identifier.

67 We shall refer to this triplet of directories by TDIR

type
64 BDIR = BI $\overrightarrow{m}$ SI-**set**
65 SDIR = SI $\overrightarrow{m}$ DI-**set**
66 DDIR = DI $\overrightarrow{m}$ RoDD
67 TDIR = BDIR × SDIR × DDIR
value
64 attr_BDIR: CP → BDIR
65 attr_SDIR: CP → SDIR
66 attr_DDIR: CP → DPL

All three directories are *programmable attributes*.

The business swarm concept can be visualized by grouping together drones of the same swarm in the visualizarion of the aggregate set of enterprise drones. Figure 3 attempts this visualization.

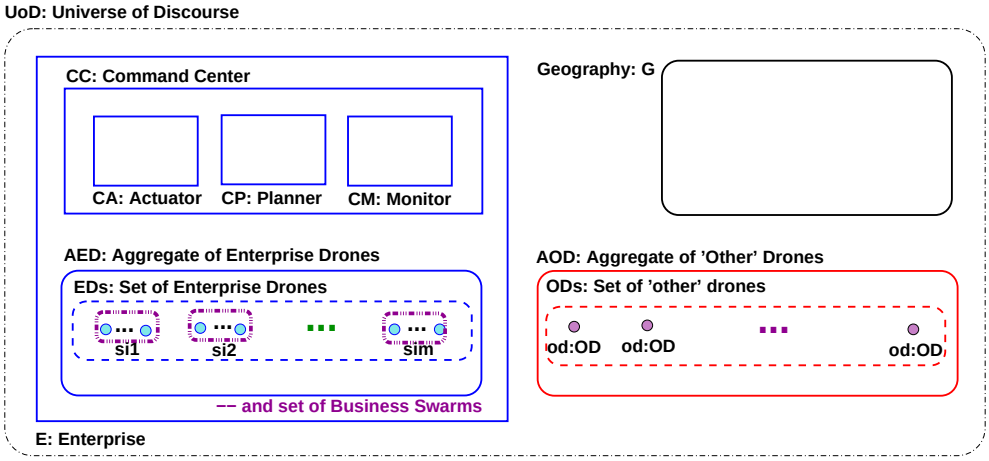

Figure 3: Conceptual Swarms of the Universe of Discourse

68 For the planners of all universes of discourse the following must be the case.

(a) The swarm directory must
   i have entries for exactly the swarms of the business directory,
   ii define disjoint sets of enterprise drone identifiers, and
   iii these sets must together cover all enterprise drones.

(b) The drone directory must record the present position, the past positions, a list, dpl:DPL, and, besides satisfying axioms 61, satisfy some further constraints:

i they must list exactly the drone identifiers of the aggregate of enterprise drones, and the sum total of its enterprise drone identifiers must be exactly those of the enterprise drones aggregate of enterprise swarms, and

ii the head of a drone's present and past position list must similarly be within reasonable distance of that drone's current position.

axiom
68 ∀ uod:UpD,e:E,cp:CP,g:G •
68 e=obs_E(uod)∧cp=obs_CP(obs_CC(e)) ⇒
68(a) **let** (bdir,sdir,ddir) = (attr_BDIR,attr_SDIR,attr_DDIR)(cp) **in**
68(a)i ∪ **rng** bdir = **dom** sdir
68(a)ii ∧ ∀ si,si′SI•{si,si′}⊆**dom** sdir∧si≠si′ ⇒
68(a)ii sdir(s) ∩ sdir(s′) = {}
68(a)iii ∧ ∪ **rng** sdir = xtr_dis(e)
68(b)i ∧ **dom** ddir = xtr_dis(e)
68(b)ii ∧ ∀ di:DI•di ∈ **dom** ddir
68(b)ii **let** (d,dpl) = (attr_DDIR(cp))(di) **in**
68(b)ii dpl ≠ ⟨⟩
68(b)ii ⇒ neighbours(f,**hd**(dpl))
68(b)ii ∧ neighbours(**hd**(dpl),
68(b)ii attr_EDPOS(xtr_D(obs_Ss(e))(di)))
68 **end end**

2.4.10 Actuator Attributes

The actuator receives, from the planner, flight directives as to which enterprise drones should be redirected. The actuator maintains a record of most recent and all past such flight directives. Finally, the actuator, effects the directives by informing designated enterprise drones as to their next flight plans.

69 Actuators have one programmable attribute: a flight directive directory. It lists, for each enterprise drone, by identifier, a pair: its current flight plan and a list of past flight plans.

type
69 FDDIR = EDI $\overrightarrow{m}$ (FP × FP*)
value
69 attr_FDDIR: CA → FDDIR

2.4.11 Geography Attributes

Constituent Types: The constituent types of *longitude, latitude* and *altitude* and *positions*, of a *geography*, were introduced in Items 3.

70 A further concept of geography is that of *area*.

71 An area, a:A, is a subset of positions within the geography.

type
70 A = P**-infset**
axiom
71 ∀ uod:UoD,g:G,a:A• g=obs_G(uod) ⇒ a ⊆ attr_EPS(g)

Attributes

72 Geographies have, as one of their attributes, a *Euclidian Point Space*, in this case, a *compact*[25] infinite set of three-dimensional positions.

type
72 EPS = P**-infset**
value
72 attr_EPS: G → EPS

Further geography attributes reflect the "lay of the land and the weather right now!".

73 The "lay of the land", L is a "conglomerate" further undefined geodetics and cadestra[26]

74 The "weather"W is another "conglomerate" of temperature, humidity, precipitation, air pressure, etc.

type
73 L
74 W
value
73 attr_L: G → L
74 attr_W: G → W

3 Operations on Universe of Discourse States

Before we analyse & describe perdurants let us take a careful look at the actions that drone and swarm behaviours may take. We refer to this preparatory analysis & description as one of analysing & describing the state operations. From this analysis & description we move on to the analysis & description of behaviours, events and actions. The idea is to be able to prove some relations between the two analyses & descriptions: the state operation and the behaviour analyses & descriptions. We refer to [26, Sects. 2.3 and 2.5].

3.1 The Notion of a State

A state is any subset of parts each of which contains one or more dynamic attributes. Following are examples of states of the present case study: a space of interest, an aggregate of 'business' swarms, an aggregate of 'other' swarms, a pair of the aggregates just mentioned, a swarm, or a drone.

[25] In mathematics, and more specifically in general topology, compactness is a property that generalizes the notion of a subset of Euclidean space being closed (that is, containing all its limit points) and bounded (that is, having all its points lie within some fixed distance of each other). Examples include a closed interval, a rectangle, or a finite set of points.

[26] land surface altitude, streets, buildings (tall or not so tall), power lines, etc.

3.2 Constants

Some quantities of a given universe of discourse are constants. Examples are the unique identifiers of the:

75 enterprise, e_i;
76 aggregate of 'other' drones, oi;
77 geography, g_i;
78 command center, cc_i;
79 monitor, cm_i;
80 planner, cp_i;
81 actuator, ca_i;
82 set of 'other' drones, od_{is};
83 set of enterprise drones, ed_{is};
84 and the set of all drones, ad_{is}.

value
75 aed_i:EI = uid_AED(obs_AED(uod))
76 aod_i:OI = uid_AOD(obs_AOD(uod))
77 g_i:GI = uid_G(obs_G(uod))
78 cc_i:CCI = uid_CC(obs_CC(obs_AED(uod)))
79 cm_i:CMI = uid_CM(obs_CM(obs_CC(obs_AED(uod))))
80 cp_i:CPI = uid_CP(obs_CP(obs_CC(obs_AED(uod))))
81 ca_i:CAI = uid_CA(obs_CA(obs_CC(obs_AED(uod))))
82 od_{is}:ODIs = xtr_dis(obs_AOD(uod))
83 ed_{is}:EDIs = xtr_dis(obs_AED(uod))
84 ad_{is}:DI-**set** = $od_{is} \cup ed_{is}$

3.3 Operations

An operation is a function from states to states. Following are examples of operations of the present case study: a drone *transfer:* leaving a swarm to join another swarm, a drone *changing course:* an enterprise drone changing course, a swarm *split:* a swarm splitting into two swarms, and swarm *join:* two swarms joining to form one swarm.

3.3.1 A Drone Transfer

85 The *transfer* operator specifies two distinct and unique identifiers, si, si′, of two enterprise swarms, and the unique identifier, di, of an enterprise drone – all of the same univserse of discourse. The *transfer* operation further takes a universe of discourse and yields a universe of discourse as follows:

86 The input argument 'from' and 'to' swarm identifiers are diffent.

87 The initial and the final state aggregates of enterprise drones, 'other' drones and geographies are unchanged.

88 The initial and final state monitors and actuators are unchanged.

89 The business and the drone directors of the initial and final planner are unchanged.

90 The 'from' and 'to' input argument swarm identifiers are in the swarm directory and the input argument drone identifiers is in the initial swarm directory entry for the 'from' swarm identifier.

91 The input argument drone identifier is in final the swarm directory entry for the 'to' swarm identifier.

92 And the final swarm directory is updated ...

```
value
85  transfer: DI × SI × SI → UoD ~→ UoD
85  transfer(di,fsi,tsi)(uod) as uod′
86      fsi ≠ tsi ∧
85      let aed = obs_AED(uod), aed′ = obs_AED(uod′), g = obs_G(uod), g′ = obs_G(uod′) in
85      let cc = obs_CC(aed), cc′ = obs_CC(aed′), aod = obs_AOD(uod), aod′ = obs_AOD(uod′) in
85      let cm = obs_CM(cc), cm′ = obs_CM(cc′), cp = obs_CP(cc), cp′ = obs_CP(cc′) in
85      let ca = obs_CA(cc), ca′ = obs_CA(cc′) in
85      let bdir = attr_BDIR(cc), bdir′ = attr_BDIR(cc′),
85          sdir = attr_SDIR(cc), sdir′ = attr_SDIR(cc′),
85          ddir = attr_DDIR(cc), ddir′ = attr_DDIR(cc′) in
87      post: aed = aed′ ∧ aod = aod′ ∧ g = g′ ∧
88            cm = cm′ ∧ ca = ca′ ∧
89            bdir = bdir′ ∧ ddir = ddir′
90      pre {fsi,tsi} ⊆ dom sdir ∧ di ∈ sdir(fsi)
91      post di ∉ sdir(fsi′) ∧ di ∈ sdir(tsi′) ∧
92           sdir′ = sdir † [fsi↦sdir(fsi)∪ di] † [tsi↦sdir(tsi)\di]
85      end end end end end
```

3.3.2 Etcetera !

Similar expositions of "machine state"-oriented operational semantics can be give for the actions illustrated in Sects. 4.4.1 –4.4.6.

4 Perdurants

We observe that the term *train* can have the following "meanings": the *train*, as an *endurant*, parked at the railway station platform, i.e., as a *composite part;* the *train*, as a *perdurant*, as it "speeds" down the railway track, i.e., as a *behaviour;* the *train*, as an *attribute*. This observation motivates that we "magically", as it were, introduce a **compile**r function, cf. [30, Sect. 4]

4.1 System Compilation

The **compile**r function "worms" its way, so-to-speak, "down" the "hierarchy" of parts, from the universe of discourse, via its immediate sup-parts, and from these to their sub-parts, and so on, until the **compile**r reaches atomic parts. We shall henceforth do likewise.

4.1.1 The Compile Functions

93 Compilation of a *universe of discourse* results in

(a) the `RSL Text` of the *core* of the universe of discourse behaviour (which we set to **skip** – allowing us to ignore *core* arguments),

(b) followed by the `RSL Text` of the parallel composition of the compilation of the enterprise,

(c) followed by the `RSL Text` of the parallel composition of the compilation of the geography,

(d) followed by the `RSL Text` of the parallel composition of the compilation of the aggregate of 'other' drones.

93 **compile**$_{UoD}$(uod) ≡
93(a) $\mathcal{M}_{\text{uid\_UoD(uod)}}$(mereo_UoD(uod),sta(uod))(pro(uod))
93(b) ‖ **compile**$_{AED}$(obs_AED(uod))
93(c) ‖ **compile**$_{G}$(obs_G(uod))
93(d) ‖ **compile**$_{AOD}$(obs_AOD(uod))

94 Compilation of an *enterprise* results in

(a) the `RSL Text` of the *core* of the enterprise behaviour (which we set to **skip** – allowing us to ignore *core* arguments),

(b) followed by the `RSL Text` of the parallel composition of the compilation of the enterprise aggregate of enterprise drones,

(c) followed by the `RSL Text` of the parallel composition of the compilation of the enterprise command center.

94 **compile**$_{AED}$(e) ≡
94(a) $\mathcal{M}_{\text{uid\_AED(e)}}$(mereo_E(e),sta(e))(pro(e))
94(b) ‖ **compile**$_{EDs}$(obs_EDs(e))
94(c) ‖ **compile**$_{CC}$(obs_CC(e))

95 Compilation of an *enterprise aggregate of enterprise drones* results in

(a) the `RSL Text` of the *core* of the aggregate behaviour (which we set to **skip** – allowing us to ignore *core* arguments),

(b) followed by the `RSL Text` of the parallel composition of the distributed compilation of the enterprise aggregate's set of enterprise drones.

95 **compile**$_{EDs}$(es) ≡
95(a) $\mathcal{M}_{\text{uid\_EDs(es)}}$(mereo_EDS(es),sta(es))(pro(es))
95(b) ‖ {**compile**$_{ED}$(ed)|ed:ED•ed ∈ obs_EDs(s)}

96 Compilation of an *enterprise drone* results in

(a) the `RSL Text` of the *core* of the enterprise drone behaviour – which is what we really wish to express – and since enterprise drones are here considered atomic, that is where the compilation of enterprise ends.

96 **compile**$_{ED}$(ed) ≡
96(a) $\mathcal{M}_{\text{uid\_ED(ed)}}$(mereo_ED(ed),sta(ed))(pro(ed))

97 Compilation of an *aggregate of 'other' drones* results in

(a) the `RSL Text` of the *core* of the aggregate 'other' drones behaviour (which we set to **skip** – allowing us to ignore *core* arguments) –

(b) followed by the `RSL Text` of the parallel composition of the distributed compilation of the 'other' drones in the 'other' drones' aggregate set of 'other' drones.

97 **compile**$_{AOD}$(aod) ≡
97(a) $\mathcal{M}_{\text{uid\_OD(od)}}$(mereo_S(ods),sta(ods))(pro(ods))
97(b) ‖ {**compile**$_{OD}$(od)|od:OD•od ∈ obs_ODs(ods)}

98 Compilation of a(n) *'other' drone* results in

(a) the `RSL Text` of the *core* of the 'other' drone behaviour – which is what we really wish to express – and since 'other' drones are here considered atomic, that is where the compilation of the 'other' drones aggregate

98(a) **compile**_{OD}(ed) ≡
98(a) $\mathcal{M}_{\text{uid\_OD(od)}}$(mereo_OD(od),sta(od))(pro(od))

99 Compilation of an atomic *geography* results in

(a) the `RSL Text` of the *core* of the geography behaviour.

99 **compile**$_{G}$(g) ≡
99(a) $\mathcal{M}_{\text{uid\_G(g)}}$(mereo_G(g),sta(g))(pro(g))

100 Compilation of a composite *command center* results in

(a) the `RSL Text` of the *core* of the command center behaviour (which we set to **skip** – allowing us to ignore *core* arguments)

(b) followed by the `RSL Text` of the parallel composition of the compilation of the command monitor,

(c) followed by the `RSL Text` of the parallel composition of the compilation of the command planner,

(d) followed by the `RSL Text` of the parallel composition of the compilation of the command actuator.

100 **compile**$_M$(cc) ≡
100(a) $\mathcal{M}_{\text{uid\_CC(cc)}}$(mereo_CC(cc),sta(cc))(pro(cc))
100(b) ‖ **compile**$_{CC}$(obs_CM(cc))
100(c) ‖ **compile**$_{CP}$(obs_CP(cc))
100(d) ‖ **compile**$_{CA}$(obs_CA(cc))

101 Compilation of an atomic *command monitor* results in

(a) the `RSL Text` of the *core* of the monitor behaviour.

101 **compile**$_{CM}$(cm) ≡
101(a) $\mathcal{M}_{\text{uid\_CM(cm)}}$(mereo_CM(cm),sta(cm))(pro(cm))

102 Compilation of an atomic *command planner* results in

(a) the `RSL Text` of the *core* of the planner behaviour.

102 **compile**$_{CP}$(cp) ≡
102(a) $\mathcal{M}_{\text{uid\_CP(cp)}}$(mereo_CP(cp),sta(cp))(pro(cp))

103 Compilation of an atomic *command actuator* results in

(a) the `RSL Text` of the *core* of the actuator behaviour.

103 **compile**$_{CA}$(ca) ≡
103(a) $\mathcal{M}_{\text{uid\_CA(ca)}}$(mereo_CA(ca),sta(ca))(pro(ca))

4.1.2 Some `CSP` Expression Simplifications

We can justify the following `CSP` simplifications [41, 43, 53, 55]:

104 **skip** in parallel with any CSP expression *csp* is *csp*.

105 The distributed parallel composition of the distributed parallel composition of `CSP` expressions, csp(i,j), i indexed over I, i.e., i:I, and j:J respectively, is the distributed parallel composition over `CSP` expressions, csp(i,j), i.e., indexed over (i,j):$I\times J$ – where the index sets iset and jset are assumed.

axiom
105 **skip** ‖ csp ≡ csp
105 ‖{‖{csp(i,j)|i:I•i∈iset}|j:J•j∈jset} ≡ ‖{csp(i,j)|i:I,j:J•i∈I-**set**∧j∈J-**set**}

4.1.3 The Simplified Compilation

106 The simplified compilation results in:

106 **compile**(uod) ≡
96(a) { $\mathcal{M}_{\text{uid\_ED(ed)}}$(mereo_ED(ed),sta(ed))(pro(ed))
96(a) | ed:ED • ed ∈ xtr_Ds(obs_AED(uod)) }
98(a) ‖ { $\mathcal{M}_{\text{uid\_OD(od)}}$(mereo_OD(od),sta(od))(pro(od))
98(a) | od:OD • od ∈ xtr_ODs(obs_AOD(uod)) }
99(a) ‖ $\mathcal{M}_{\text{uid\_G(g)}}$(mereo_G(g),sta(g))(pro(g))
99(a) **where** g ≡ obs_G(uod)
101(a) ‖ $\mathcal{M}_{\text{uid\_CM(cm)}}$(mereo_CM(cm),sta(cm))(pro(cm))
101(a) **where** cm ≡ obs_CM(obs_CC(obs_E(uod)))
102(a) ‖ $\mathcal{M}_{\text{uid\_CP(cp)}}$(mereo_CP(cp),sta(cp))(pro(cp))
102(a) **where** cp ≡ obs_CP(obs_CC(obs_E(uod)))
103(a) ‖ $\mathcal{M}_{\text{uid\_CA(ca)}}$(mereo_CA(ca),sta(ca))(pro(ca))
103(a) **where** ca ≡ obs_CA(obs_CC(obs_E(uod)))

107 In Item 106's Items 96(a), 98(a), 99(a), 101(a), 102(a), and 103(a) we replace the "anonymous" behaviour names $\mathcal{M}$ by more meaningful names.

107 **compile**(uod) ≡
96(a) { enterprise_drone$_{\text{uid_ED(ed)}}$(mereo_ED(ed),sta(ed))(pro(ed))
96(a) | ed:ED • ed ∈ xtr_Ds(obs_AED(uod)) }
98(a) ‖ { other_drone$_{\text{uid_OD(od)}}$(mereo_OD(od),sta(od))(pro(od))
98(a) | od:OD • od ∈ xtr_ODs(obs_AOD(uod)) }
99(a) ‖ geography$_{\text{uid_G(g)}}$(mereo_G(g),sta(g))(pro(g))
99(a) **where** g ≡ obs_G(uod)
101(a) ‖ monitor$_{\text{uid_CM(cm)}}$(mereo_CM(cm),sta(cm))(pro(cm))
101(a) **where** cm ≡ obs_CM(obs_CC(obs_E(uod)))
102(a) ‖ planner$_{\text{uid_CP(cp)}}$(mereo_CP(cp),sta(cp))(pro(cp))
102(a) **where** cp ≡ obs_CP(obs_CC(obs_E(uod)))
103(a) ‖ actuator$_{\text{uid_CA(ca)}}$(mereo_CA(ca),sta(ca))(pro(ca))
103(a) **where** ca ≡ obs_CA(obs_CC(obs_E(uod)))

4.2 An Early Narrative on Behaviours

4.2.1 Either Endurants or Perdurants, Not Both !

First the reader should observe that the manifest parts, in some sense, do no longer "exist" ! They have all been replaced by their corresponding behaviours. These behaviours embody all the qualities of their "origin": the unique identifiers, the mereology, and all the attributes – the latter in one form or another: the static attributes as constants (`referred to in the bodies of the behaviour definitions`); the programmable attributes as arguments

(‘‘carried over’’ from one invocation to the next); and the remaining dynamic attributes as "inputs" (whose varying values are ‘‘accessed’’ through [dynamic attribute] channels).

4.2.2 Focus on Some Behaviours, Not All !

Secondly we focus, in this case study, only on the behaviour of the *planner*. The other behaviours, the *'other' drones*, *enterprise drones*, *monitor*, *actuator*, and the *geography*, are, in this case study of less interest to us. That is, other case studies could focus on the behaviours of *drones*, or *geographies*, or *monitor*, or *actuator*.

4.2.3 The Behaviours – a First Narrative

*Drone*s "continuously" offer their identified dynamics (location, velocity, and possibly more) **to** the *monitor*. *Enterprise drones* "continuously", and in addition, offers to accept flight guidance **from** the *actuator*. The *monitor* "continuously sweeps" the air space and collects the identities of all recognizable drones and their dynamics, and offers this **to** the *planner*. The *planner* does all the interesting work ! It effects the *allocation/reallocation* of drones to/from business swarms; it *calculates enterprise drone flights* and instructs the *actuator* to offer such flight plans to relevant drones; etcetera ! Finally the *actuator*, as instructed by the *planner*, offers flight guidance, as per instructions **from** the *planner*, **to** all or some *enterprise drones*.

4.3 Channels

Channels is a concept of CSP [41, 42, 43].

CSP channels are a means for synchronising behaviours and for communicating values between synchronised behaviours, as well as, as a technicality, conveying values of most kinds of dynamic attributes of parts (i.e., endurants) to "their" behavioural counterparts.

There are thus two starting point for the analysis & description of channels: the mereologies and the dynamic attributes of parts. Here we shall single out the following parts and behaviours: the command *monitor, planner* and *actuator*, the *enterprise drones* and the *'other' drones*, and the *geography*. We refer to Fig. 4 on the next page, a slight "refinement" of Fig. 1 on Page 6.

4.3.1 The Part Channels

General Remarks: Let there be given a *universe of discourse*. Let us analyse the *unique identifiers* and the *mereologies* of the *planner* cp: (cpi,cpm), *monitor* cm: (cmi,cmm) and *geography* g: (mi,mm), where cpm = (cai,cmi,gi), cmm = ($\{di_1, di_2, \ldots, di_n\}$,cpi) and gm = (cpi,$\{di_1, di_2, \ldots, di_n\}$).

We now interpret these facts. When the *planner mereology* specifies the unique identifiers of the *actuator*, the *monitor*, and the *geography*, then that shall mean there there is a way of communicating messages between the actuator, and the geography, amd one side, and the plannner on the other side.

108 We shall therefore, in a first step of specification development, think of a "grand" array channel over which all communication between behaviours take place. See Fig. 4 on the next page.

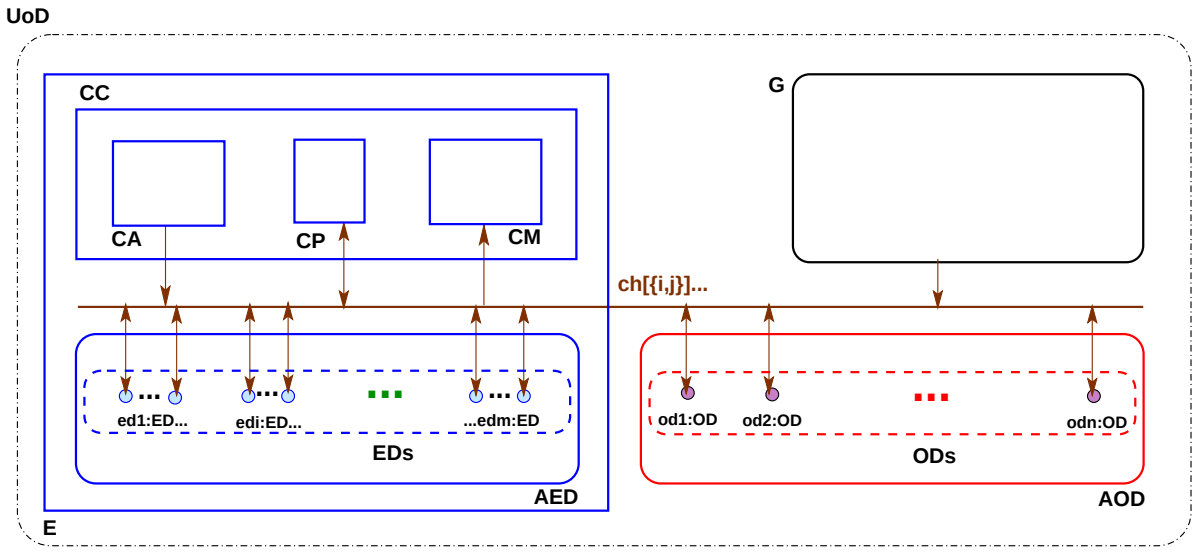

Figure 4: Universe of Discourse with General Channel: **ch[{i,j}] ...**

109 Example indexes into this array channel are shown in the formulas just below.

type
108 MSG
channel
108 {ch[fui,tui]|fui,tui:PI • ...}:MSG
value
109 ch[cpi,cai]!msg *output from* planner *to* actuator.
109 ch[cpi,cai]? *input from* planner *to* actuator.
109 etc.

We presently leave the type of messages, MSG, that can be communicated over this "grand" channel further unspecified. We also leave unspecified the pair of distinct unique identifiers that index the channel array. We emphasize that the uniqueness of all part identifiers allow us to use pairs of such as indices. Expression ch[fui,tui]!,sg thus expresses *output from* behaviour indexed by fuit *to* behaviour indexed by tui, whereas expression ch[tui,fui]? thus expresses *input from* behaviour indexed by tui *to* behaviour indexed by fui. Not all combinations of unique identifiers are needed. The channel array is "sparse" ! That property allows us to refine the "grand" channel into the channels illustrated on Fig. 5 on the facing page. Some channels are array channels: The channels to the drones whether all drones, or just the enterprise drones. Other channels are "single" channels: these are the channels which are anchored in parts with a priori known, i.e., constant unique identifiers.

Part Channel Specifics

110 There is an array channel, d_cm_ch[di,cm_i]:D_CM_MSG, from any *drone* ([di]) behaviour to the *monitor* behaviour (whose unique identifier is cm_i). The channel, as an array, forwards the current drone dynamics D_CM_MSG = CuDD.

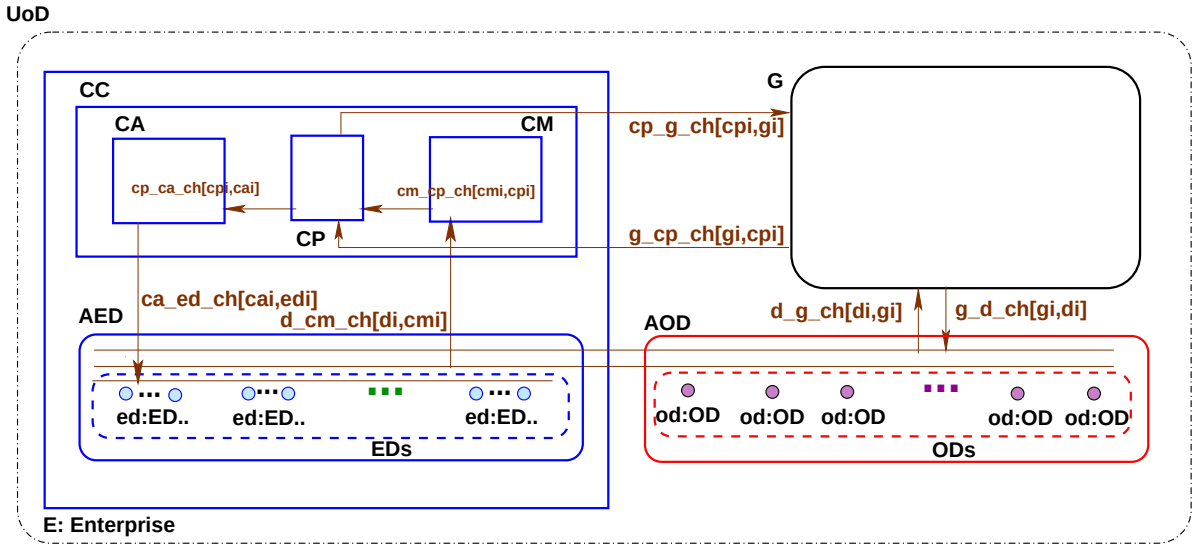

Figure 5: Universe of Discourse with Specific Channels

type
110 D_CM_MSG = CuDD
channel
110 {d_cm_ch[di,cm_i]|di:(EDI|ODI)•di ∈ dis}:D_CM_MSG

111 There is a channel, cm_cp_ch[cm_i,cp_i, from the *monitor* behaviour (cm_i) to the *planner* behaviour (cp_i). It forwards the monitor's records of drone dynamics CM_CP_MSG = MRoDD.

type
111 CM_CP_MSG = MRoDD

111 **channel** m_cp_ch[cm_i,cp_i]:CM_CP_MSG

112 There is a channel, cp_ca_ch[cp_i,ca_i]:CP_CA_MSG, from the *plannner* behaviour (cp_i) to the *actuator* behaviour (ca_i). It forwards flight plans CP_CA_MSG = FP.

type
112 CP_CA_MSG = EID $\overrightarrow{m}$ FP
channel
112 cp_ca_ch[cp_i,ca_i]:CM_CP_MSG

113 There is an array channel, ca_ed_ch[cai,edi], from the *actuator behaviour* (ca_i) to the *enterprise drone* behaviours (edi for suitable edis). It forwards flight plans, CA_ED_MSG = FP, to enterprise drones in a designated set.

type
113 CA_ED_MSG = EID × FP
channel
113 {ca_ed_ch[*cai*,edi]|edi:EDI•edi ∈ *edis*}:CA_ED_MSG

114 There is an array channel, g_d_ch[di,g_i]:D_G_MSG, from all the *drone* behaviours (di) to the *geography* behaviour The channels convey, requests for an *immediate geography* for and around a *point*: D_G_MSG = P.

type
114 D_G_MSG = P
channel
114 {d_g_ch[di,g_i]|di:(EDI|ODI)•di ∈ *dis*}:D_H_MSG

115 There is an array channel, g_d_ch[g_i,di]:G_D_MSG, from the *geography* behaviour to all the *drone* behaviours. The channels convey, for a requested *point*, the immediate geography for that area: G_D_MSG = ImG.

type
115 G_D_MSG = ImG
channel
115 {g_d_ch[g_i,di]|di:(EDI|ODI)•di ∈ *dis*}:G_D_MSG

4.3.2 Attribute Channels, General Princiles

Some of the drone attributes are *reactive*. Being reactive means that their values change surreptitiously. In the physical world of parts that means that these vales must be measured, or somehow ascertained, whenever needed, i.e., "on the fly". Now "our world" is that of a domain description. When dealing with endurants, the value of an attribute, a:A, of part p:P, is expressed as attr_A(p). When dealing with perdurants, that same value is to be expressed as attr_A_ch[uid_P(p)] ?.

116 This means that we must declare a channel for each part with one or more *dynamic*, however not including *programmable*, attributes A1, A2, ..., An.

channel
116 attr_A1_ch[p_i]:A1, attr_A2_ch[p_i]:A2, ..., attr_An_ch[p_i]:An

117 If there are several parts, p1,p2,...,pm:P then an array channel over indices $p1_i$,$p2_i$,...,pm_i is declared for each applicable attribute.

channel
117 {attr_A1_ch[pj_i]|pj_i:PI•pj_i∈{$p1_i$,$p2_i$,...,pm_i}}:A1,
117 {attr_A2_ch[pj_i]|pj_i:PI•pj_i∈{$p1_i$,$p2_i$,...,pm_i}}:A2,
117 ...
117 {attr_An_ch[pj_i]|pj_i:PI•pj_i∈{$p1_i$,$p2_i$,...,pm_i}}:An

4.3.3 The Case Study Attribute Channels

'Other' Drones: 'Other' drones have the following not biddable or programmable dynamic channels:

118 dynamics, including velocity, acceleration, orientation and position, {attr_DYN_ch[odi]:DYN|odi:ODI•odi∈*odis*}.

channel
118 {attr_DYN_ch[odi]:DYN|odi:ODI•odi ∈ *odis*}

Enterprise Drones: Enterprise drones have the following not biddable or programmable dynamic channels:

119 dynamics, including velocity, acceleration, orientation and position, {attr_DYN_ch[edi]:DYN|edi:EDI•edi∈*odis*}.

channel
119 {attr_DYN_ch[odi]:DYN|odi:ODI•odi ∈ *odis*}

Geography: The geography has the following not biddable or programmable dynamic channels:

120 land, attr_L_ch[g_i]:L, and

121 weather, attr_W_ch[g_i]:W.

channel
120 attr_L_ch[g_i]:L
121 attr_W_ch[g_i]:W

We do not show any graphics for the attribute channels.

4.4 The Atomic Behaviours

TO BE WRITTEN

4.4.1 Monitor Behaviour

122 The signature of the monitor behaviour

(a) lists the monitor's unique identifier, carries the monitor's mereology, has no static arguments (... maybe ...), has the programmable time-stamped recordings, dtp, of all drone positions (present and past) and

(b) further designates the **in**put channel d_cm_ch[*.*] from all drones and the channel **out**put cm_cp_ch[cmi,cpi] to the planner.

123 The monitor [otherwise] behaves as follows:

(a) All drones provide as input, d_cm_ch[di,cmi]?, their time-stamped positions, rec.

(b) The programmable mrodd attribute is updated, mrodd′, to reflect the latest time stamped dynamics per drone identifier.

(c) The updated attribute is is provided to the planner.

(d) Then the monitor resumes being the monitor, forwarding, as the progammable attribute, the time-stamped drone position recording.

```
value
122(a)  monitor: cmi:CMI×cmm:(dis:DI-set×cpi:CPI) → MRoDD →
122(b)     in {d_cm_ch[ di,cmi ]|di:DI•di∈dis} out cm_cp_ch  Unit
123     monitor(mi,(dis,cpi))(mrodd) ≡
123(a)     let rec = {[ di ↦ d_cm_ch[ di,cmi ]?|di:DI•di∈dis ]} in
123(b)     let mrodd′ = mrodd † [ di↦⟨rec(di)⟩^mrodd(di)|di:DI•di∈dis ] in
123(c)     cm_cp_ch[ cmi,cpi ] ! mrodd′;
123(d)     monitor(cmi,(dis,cpi))(mrodd′)
123        end end
123  axiom cmi=cm_i∧cpi=cp_i
```

We have decided to let the monitor maintain the present and past time-stamped drone positions. It is the monitor which records these positions. Not the planner. But the monitor provides these traces, again-and-again, to the planner.

4.4.2 Planner Behaviour

124 The signature of the planner behaviour

(a) lists the planner's unique identifier, carries the planner's mereology, has, perhaps ..., some static arguments, has the programmable planner directories, and

(b) further designates the single **in**put channel cm_cp_ch and the single **out**put channel cp_ca_ch.

125 The planner [otherwise] behaves as follows:

(a) the planner [internal] non-deterministically ("coin-flipping") decides whether to transfer a drone between business swarms, or to calculate flight plans, or ... other.

(b) Depending on the [outcome of the "coin-flipping"] the planner

(c) either effects a transfer,

i by delegating to an auxiliary function, transfer, the necessary modifications of the swarm directory –

ii whereupon the planner behaviour resumes;

(d) or effects a [re-]calculation on drone flights,

i by, again, delegating to an auxiliary function, flight_planning, the necessary calculations –

ii which are communicated to the *actuator*,

iii whereupon the planner behaviour resumes;

(e) or ... other !

```
value
125  planner: cpi:CPI × (cai×CAI×cmi:CMI×gi:GI) × TDIR →
125     in cm_cp_ch[ cmi,cpi ], g_cp_ch[ gi,cpi ] out cp_ca_ch[ cpi,cai ]  Unit
125  planner(cpi,(cai,cmi,gi),...)(bdir,sdir,ddir) ≡
125(a)     let cmd = "transfer" ⌈⌉ "flight_plan" ⌈⌉ ... in
125(b)     cases cmd of
125(c)        "transfer" →
125(c)i           let sdir′ = transfer(tdir) in
125(c)ii           planner(cpi,(cai,cmi,gi),...)(bdir,sdir′,ddir) end
125(d)        "flight_plan" →
125(d)i           let ddir′ = flight_planning(tdir) in
125(d)ii           planner(cpi,(cai,cmi,gi),...)(bdir,sdir,ddir′) end
125(e)        ...
125     end
125  axiom cpi=cp_i∧cai=ca_i∧cmi=cm_i∧gi=g_i
```

The Auxiliary transfer Function

126 The *transfer* function has a simpler signature than the planner behaviour in that it need not communicate with other behaviours.

(a) The transfer function *internal non-deterministically chooses* a business designator, bi;

(b) from among that business' swarm designators it *internal non-deterministically chooses* two distinct swarm designators, fsi,tsi;

(c) and from the fsi entry in sdir (which is set of enterprise drone identifiers), it *internal non-deterministically chooses* an enterprise drone identifier, di.

(d) Given the swarm and drone identifiers *the resulting swarm directory* can now be made to reflect the transfer: reference to di is *removed* from the fsi entry in sdir and that reference instead *inserted* into the tsi entry.

```
value
126  transfer:  TDIR → SDIR
126  transfer(bdir,sdir,ddir) ≡
126(a)     let bi:BI•bi ∈ dom bdir in
126(b)     let fsi,tsi:SI•{fsi,tsi}⊆bdir(bi)∧fsi≠tsi in
126(c)     let di:DI•di ∈ sdir(fsi) in
126(d)     sdir † [ fsi↦sdir(fsi)\{di} ] † [ tsi↦sdir(tsi)∪{di} ]
126     end end end
```

The Auxiliary flight_planning Function

127 The signature of the flight_planning behaviour needs two elements: the triplet of business, swarm and drone directories, and the planner-to-actuator channel.

(a) The flight_planning behaviour offers to accept the time-stamped recordings of the most recent drone positions and dynamics as well as all the past such recordings.

(b) The flight_planning behaviour selects, *internal, non-deterministically* a business, designated by bi,

(c) one of whose swarms, designated by si, it has thus decided to perform a flight [re-]calculation for.

(d) An objective for the new flight plan is chosen.

(e) The flight_plan is calculated.

(f) That flight plan is communicated to the *actuator*.

(g) And the flight plan, appended to the drone directory's (past) flight plans.

```
value
127  flight_planning: TDIR → in cm_cp_ch[ cm_i,cp_i ], out cp_ca_ch[ cp_i,ca_i ]  DTP
127  flight_planning(bdir,sdir,ddir) ≡
127(a)      let dtp = cm_cp_ch[ cp_i,ca_i ] ? ,
127(b)          bi:BI • bi ∈ dom bdir
127(c)      let si:SI • si ∈ bdir(bi) in
127(d)      let fp_obj:fp_objective(bi,si) in
127(e)      let flight_plan = calculate_flight_plan(dtp,sdir(si),fp_obj,tdir) in
127(f)      cp_ca_ch[ cp_i,ca_i ] ! flight_plan ;
127(g)      ⟨flight_pla⟩^ddir
127       end end end end
type
127(d)   FP_OBJ
value
127(d)   fp_objective: BI × SI → FP_OBJ
127(d)   fp_objective(bi,si) ≡ ...
```

128 The calculate_flight_plan function is the absolute focal point of the *planner*.

```
128  calculate_flight_plan: DTP × DI-set × FP−OBJ × TDIR → FP
128  calculate_flight_plan(dtp,sdir(si),fp_obj,tdir) ≡ ...
```

There are many ways of calculating flight plans. [46, Mehmood et al., Stony Brook, 2018: *Declarative vs Rule-based Control for Flocking Dynamics*] is one such:

MORE TO COME

In [50, 51, 52, Craig Reynolds: OpenSteer, *Steering Behaviours for Autonomous Characters*]

MORE TO COME

In [48, Reza Olfati-Saber: Flocking for Multi-agent Dynamic Systems: Algorithms and Theory, 2006]

MORE TO COME

The calculate_flight_plan function, Item 128 on the preceding page , is deliberately provided with all such information that can be gathered and hence can be the only 'external'[27] data that can be provided to such calculation functions,[28] and is therefore left further unspecified; future work[29] will show whether this assumption holds. If it does, then, OK, and we can proceed. If it does not, we shall revise the present model.

4.4.3 Actuator Behaviour

129 The actuator accepts a current flight plan, cfp:CFP, i.e., a number of enterprise drone identifier-indexed flight plans, from the planner.

130 The signature of the actuator behaviour lists the actuator's unique identifier, carries the actuator's mereology, has, perhaps ..., some static arguments, has the programmable flight directory, and further designates the **in**put channel cp_ca_ch[cpi,cai] and the **out**put channel ca_ed_ch[cai,*].

131 The actuator further behaves as follows:

(a) It offers to accept a current flight plan from the planner.

(b) It then proceeds to offer those enterprise drones which are designated in the flight plan their flight plan.

(c) Whereupon the actuator resumes being the actuator, now with its programmable flight plan directory updated with the latest such!

type
129 CFP = EDI $\overrightarrow{m}$ FP
value
130 actuator: cai:CAI × (cpi:CPI×edis:EDI-**set**) → FDDIR →
130 **in** cp_ca_ch[cpi,cai] **out** {ca_ed_ch[cai,edi]|edi:EDI•edi ∈ edis} **Unit**
131 actuator(cai,(cpi,edis),...)(pfp,pfpl) ≡
131(a) **let** cfp = ca_cp_ch[cai,cpi] ? **in comment:** fp:EDI $\overrightarrow{m}$ FP
131(b) ‖ {ca_ed_ch[cai,edi]!cfp(edi)|edi:EDI•edi ∈ **dom** cfp} ;
131(c) actuator(cai,(cpi,edis),...)(cfp,⟨pfp⟩^pfpl)
129 **end**

130 **axiom** cai=ca_i∧cpi=cp_i

[27] Flight plan *objectives* are here referred to as 'internal'.
[28] Well – better check this!
[29] – for you ShaoFa!

4.4.4 'Other' Drone Behaviour

132 The signature of the *'other' drone* behaviour

(a) lists the 'other' drone's unique identifier, the 'other' drone's mereology, has, perhaps ..., some static arguments; then the programmable attribute of the geography (i.e., the area, the land and the weather) it is moving over and in;

(b) then, as **in**put channels, the *inert, active, autonomous* and *biddable* attributes: velocity, acceleration, orientation and position, and, finally

(c) further designates the array **in**put channel g_d_ch[*] from the *geography* and the array **out**put channel d_cm_ch[*] to the *monitor*.

133 The 'other' drone otherwise behaves as follows:

134 internal, non-deterministically the 'other' drone chooses to either ..., or "pro"viding to the monitors request for drone "dyn"amics, or ... .

135 If the choice is ... ,

136 If the choice is "provide dynamics" the behaviour drone_monitor is invoked, with arguments similar to that of other_drone, but "marked" with an additional, "frontal" argument: "other", and with "tail", programmable arguments (⟨⟩,⟨⟩).

137 If the choice is ... .

```
value
132  other_drone: odi:ODI × (cmi:CMI×gi:GI) × ... → (DYN×ImG) →
132(b)       in attr_DYN_ch[odi],g_d_ch[gi,odi] out d_cm_ch[odi,cmi] Unit
133  other_drone(odi,(cmi,gi),...)(dyn:(v,a,o,p),img) ≡
134     let mode = "..." ⌈⌉ "pro_dyn" ⌈⌉ "..." in
134     case mode of
135        "..." → ... ,
136        "pro_dyn" → drone_moni(odi,(cmi,gi),...)(dyn:(v,a,o,p),img)
138        "..." → ...
134     end
132     end
```

138 If the choice is "provide dynamics"

(a) then the drone-monitor behaviour ascertains its dynamics (velocity, acceleration, orientation and position),

(b) informs the monitor 'thereof', and

(c) resumes being the 'other' drone with that updated, programmable dynamics.

```
value
138   drone_moni: odi:ODI × (cmi:CMI×gi:GI) × ... → (DYN×ImG) →
138       in attr_DYN_ch[odi],g_d_ch[gi,odi] out d_cm_ch[odi,cmi] Unit
138   drone_moni(odi,(cmi,gi),...)(dyn:(v,a,o,p),img) ≡
```

```
138(a)  let (ti,dyn′,img′) =
138(a)      (time(),
138(a)       (let (v′,a′,o′,p′) = attr_DYN[odi]? in
138(a)       (v′,a′,o′,p′),
138(a)        d_g_ch[odi,gi]!p′ ; g_d_ch[gi,odi]? end)) in
138(b)  d_cm_ch[odi,cmi] ! (ti,dyn′) ;
138(c)  other_drone(cai,(cpi,edis),...)(dyn′,img′)
138(a)  end
```

4.4.5 Enterprise Drone Behaviour

139 The enterprise donor lists its enterprise drone's unique identifier, carries it's mereology, has, perhaps ..., some static arguments, the programmable enterprise drone attributes: a pair of the present flight plan, and the past flight plans, and a pair of the most recently observed dynamics and immediate geography, and further designates the single **in**put channel and the **out**put channel array .

Enterprise drones otherwise behave as follows:

140 internal, non-deterministically an enterprise drone chooses to either "rec"ording the "geo"graphy, i.e., the area, land and weather it is situated in, or "pro"viding to the monitors request for drone "dyn"amics, or "acc"epting the actuators offer of a new "f"light "p"lan, or "move" "on" (i.e., continue to fly), either "follow"ing the "flight plan" most recently received from the actuator, or, "ignor"ing this directive, "just plondering on" !

141 If the choice is "rec_geo" then the enterprise_geo behaviour is invoked,

142 If the choice is "pro_dyn" (provide dynamics to the *monitor*) then the enterprise_moni behaviour is invoked,

143 If the choice is "acc_fp" then the enterprise_accept_flight_plan behaviour is invoked,

144 If the choice is "move_on" then the enterprise drone decides either to "ignore" the flight plan, or to "follow" it.

(a) If it "ignore"s the flight plan then the enterprise_ignore behaviour is invoked,

(b) If the choice is "follow" then the enterprise_follow behaviour is invoked.

```
139  enterprise_drone: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139      ((FPL×PFPL)×(DDYN×ImG)) →
139      in attr_DYN_ch[edi],g_d_ch[gi,edi],ca_ed_ch[cai,edi]
139      out d_cm_ch[edi,cmi],d_g_ch[edi,gi]  Unit
139  enterprise_drone(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img)) ≡
140      let mode = ″rec_geo″ ⌈⌉ ″pro_dyn″ ⌈⌉ ″acc_fp″ ⌈⌉ ″move_on″ in
140      case mode of
145          ″rec_geo″ → enterprise_geo(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
146          ″pro_dyn″ → enterprise_moni(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
```

```
147          ″acc_fp″ → enterprise_acc_fl_pl(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
150          ″move_on″ →
150              let m_o_mode = ″ignore″ ⌈⌉ ″follow″ in
150              case m_o_mode of
144(a)                ″ignore″ → enterprise_ignore(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
144(b)                ″follow″ → enterprise_follow(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
150              end
150              end
140      end
140      end
139   axiom  cmi=cm_i∧cai=ca_i∧gi=g_i
```

145 If the choice is "rec_geo"

(a) then dynamics is ascertained so as to obtain a positions;

(b) that position is used in order to obtain a "fresh" immediate geography;

(c) with which to resume the enterprise drone behaviour.

```
139   enterprise_geography: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139      ((FPL×PFPL)×(DDYN×ImG)) →
139      in attr_DYN_ch[edi],g_d_ch[gi,edi],ca_ed_ch[cai,edi]
139      out d_cm_ch[edi,cmi],d_g_ch[edi,gi]  Unit
139   enterprise_geography(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
145(a)    let (v,a,o,p) = attr_DYN_ch[edi]? in
145(b)    let img′ = d_g_ch[edi,gi]!p;g_d_ch[gi,edi]? in
145(c)    enterprise_drone(edi,(cmi,cai,gi),...)((fpl,pfpl),((v,a,o,p),img′))
145(a)    end end
```

146 If the choice is "pro_dyn" (provide dynamics to the *monitor*)

(a) then a triplet is obtained as follows:

(b) the current time,

(c) the dynamics (v,a,o,p), and

(d) the immediate geography of position p,

(e) such that the *monitor* can be given the current dynamics,

(f) and the enterprise drone behaviour is resumed with updated dynamics and immediate geography.

```
139   enterprise_monitor: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139      ((FPL×PFPL)×(DDYN×ImG)) →
139      in attr_DYN_ch[edi],g_d_ch[gi,edi],
139      out d_cm_ch[edi,cmi],d_g_ch[edi,gi]  Unit
139   enterprise_monitor(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
146(a)       let (ti,ddyn′,img′) =
```

```
146(b)              (time(),
146(c)                  (let (v,a,o,p) = attr_DYN[edi]? in
146(c)                  (v,a,o,p),
146(d)                      d_g_ch[edi,gi]!p;g_d_ch[gi,edi]? end)) in
146(e)      d_cm_ch[edi,cmi] ! (ti,ddyn′) ;
146(f)      enterprise_drone(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn′,img′))
146(a)      end
```

147 If the choice is "acc_fp"

(a) the enterprise drone offers to accept a new flight plan from the *actuator*

(b) and the enterprise drone behaviour is resumed with that flight plan now becoming the next current flight plan and whatever is left of the hitherto current flight plan appended to the past flight plan list.

```
139   enterprise_acc_fl_pl: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139     ((FPL×PFPL)×(DDYN×ImG)) → in ca_ed_ch[cai,edi] Unit
139   enterprise_axx_fl_pl(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
147(a)      let fpl′ = ca_ed_ch[cmi,edi] ? in
147(b)      enterprise_drone(edi,(cmi,cai,gi),...)(fp′,⟨fpl⟩^pfpl,(ddyn,img))
147(a)      end
```

148 If the choice is "move_on" and the enterprise drone decides to "ignore" the flight plan,

(a) then it ascertains where it might be moving with the current dynamics

(b) and then it just keeps moving on till it reaches that dynamics

(c) from about where it resumes the enterprise drone behaviour.

```
139   enterprise_ignore: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139     ((FPL×PFPL)×(DDYN×ImG)) →
139     in attr_DYN_ch[edi] out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
139   enterprise_ignore(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
148(a)      let (v′,a′,o′,p′) = increment(dyn,img) in
148(b)      while let (v″,a″,o″,p″) = attr_DYN_ch[odi]? in
148(b)              ∼close(p′,p″) end do manoeuvre(dyn,img) ; wait δt end ;
148(c)      enterprise_drone(cai,(cpi,edis),...)(fpl,pfpl,(attr_DYN_ch[odi]?,img))
148(a)      end
```

149 The manoeuvre behaviour is further unspecified. For a fixed wing aircraft it controls the *yaw*, the *roll* and the *pitch* of the aircraft, hence its flight path, by operating the *elevator, aileron, ruddr* and the *thrust* of the aircraft based on its current dynamics, weight (including aircraft fuel), meteorological conditions (winds etc.).

```
value
149   manoeuvre: DYN × ImG → Unit
149   manoeuvre(dyn,img) ≡ ...
```

The **wait** δ t is some drone constant.

150 If the choice is "move_on" and the enterprise drone decides to "follow" the flight plan,

(a) then, if the current flight plan has been exhausted, i.e., "used-up" it aborts (**chaos**[30])

(b) otherwise it ascertains where it might be moving, i.e., a next dynamics from with the current dynamics.

(c) So it then "moves along" until it has reached that dynamics –

(d) from about where it resumes the enterprise drone behaviour.

```
value
139  enterprise_follow: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
139     ((FPL×PFPL)×(DDYN×ImG)) →
139     in attr_DYN_ch[edi] out d_cm_ch[edi,cmi],d_g_ch[edi,gi]  Unit
139  enterprise_follow(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
150(a)  if fpl = ⟨⟩ then chaos else
150(b)  let (v′,a′,o′,p′) = increment(dyn,img,hd fpl) in
150(c)  while let (v″,a″,o″,p″) = attr_DYN_ch[odi]? in
150(c)        ∼close(p′,p″) end do manoeuvre(hd fpl,dyn,img) ; wait δ t end ;
150(d)  enterprise_drone(edi,(cmi,cai,gi),...)((tlfpl,pfpl),(attr_DYN_ch[odi]?,img))
150(a)  end end
```

151 The (overloaded) manoeuvre behaviour is further unspecified. For a fixed wing aircraft it controls the *yaw*, the *roll* and the *pitch* of the aircraft, hence its flight path, by operating the *elevator, aileron, ruddr* and the *thrust* of the aircraft based on its current dynamics, weight (including aircraft fuel), meteorological conditions (winds etc.).

```
value
151  manoeuvre: FPE × DYN × ImG → Unit
151  manoeuvre(fpe,dyn,img) ≡ ...
```

The **wait** δ t is some drone constant.

4.4.6 Geography Behaviour

152 The *geography* behaviour definition

(a) lists the geography behaviour's unique identifier, carries the its mereology, has the static argument of its Euclidean point space, and

(b) further designates the single **in**put channels cp_g_ch[cpi,gi] from the *planner* and d_g_ch[*,gi] from the drones and the **out**put channels g_cp_ch[gi,cpi] to the *planner* and g_d_ch[gi,*] to the *drones*.

153 The *geography* otherwise behaves as follows:

[30] **chaos** means that we simply decide not to describe what then happens!

(a) Internal, non-deterministically the geography chooses to either "resp"ond to a request from the "plan"ner.

(b) If the choice is

(c) "resp_plan"

i then the *geography* offers to accept a request from the *planner* for the *immediate geography* of an *area* "around" a *point* and

ii then the *geography* offers that information to the *planner*,

iii whereupon the geography resumes being that;

else if the choice is

(d) "resp_dron"

i then then the *geography* offers to accept a request from the *planner* for the *immediate geography* of an *area* "around" a *point* and

ii then the *geography* offers that information to the *planner*,

iii whereupon the geography resumes being that.

154 The area function takes a pair of a point and a pair of *land* and *weather* and yields an *immediate geography*.

```
value
152  geography: gi:GI × gm:(cpi:CPI×cmi:CMI×dis:DI-set) × EPS →
152(a)     in cp_g_ch[cpi,gi], d_g_ch[∗,gi]
152(b)     out g_cp_ch[gi,cpi], g_d_ch[gi,∗]  Unit
152  geography(gi,(cpi,cmi,dis),eps) ≡
153(a)     let mode = "resp_plan" ⌈⌉ "resp_dron" ⌈⌉ ... in
153(b)     case mode of
153(c)       "resp_plan" →
153(c)i          let p = cp_g_ch[cpi,gi] ? in
153(c)ii         g_cp_ch[gi,cpi] ! area(p,(attr_L_ch[gi]?,attr_W_ch[gi]?)) end
153(c)iii        geography(gi,(cpi,cmi,dis),eps)
153(d)       "resp_dron" →
153(d)i          let (p,di) = ⌈⌉{(d_g_ch[di,gi]?,di)|di:DI•di ∈ dis} in
153(d)ii         g_cp_ch[di,cpi] ! area(p,(attr_L_ch[gi]?,attr_W_ch[gi]?)) end
153(d)iii        geography(gi,(cpi,cmi,dis),eps)
152        end end
axiom
152  gi=g_i∧cpi=cp_i∧smi=cm_idis=dis
value
154  area: P × (L × W) → ImG
154  area(p,(l,w)) ≡ ...
```

5 Conclusion

A month's intensive experimental engineering work is over. It is time to take a break. It is time to review the "situation". What is that "situation"? It is that of having provided a first

domain analysis & description of a 'currently' "fashionable" domain of UVA's (drones) – in the style of a rigorous analysis & description according to the principles, techniques and tools laid down in [30].

5.1 Recent Domain Analysis & Description Case Studies

The current case study is one of several.

- *Urban Planning* [35],
- *Documents* [25],
- *Credit Cards* [21],
- *Weather Information Systems* [24],
- *The Tokyo Stock Exchange* [28],
- *Pipelines* [17],
- *Road Transportation* [18],
- *Web & Transaction*-based Software [16],
- *"The Market"* [3],
- *Container [Shipping] Lines* [8] and
- *Railway Systems* [2, 31, 4, 49, 56].

These case studies helped refine the method for the analysis & description of manifest domains.

5.2 What Am I Not Happy About ?

There are a number of things, in the present case study, with which I am not at all that "happy" !

- *There are a number of function and behaviour definitions that could be made a little bit simpler and more elegant.*
- *And the enterprise_ignore and enterprise_follow behaviour definitions, of course so similar, should be made into one "cased" definition.*
- *The treatment of drone movements.* This treatment you will find in Items 148(b) on Page 43 and 150(c) on Page 44. I consider the use of the **while ... do ... end wait** δt a "fudge". The escape to this "fudging" is necessitated by the lack of proper means in all so-called formal specification languages[31] for including classical calculus, to with: differential equations and integrals.
- *The [absent] treatment of drone manoeuvring.* This treatment you will find in Items 148(b) on Page 43 and 149 on Page 43, and in Items 150(c) on Page 44 and 151 on Page 44.

MORE TO COME

- ...

MORE TO COME

[31] Some such formal specification languages, including "our own", are: `Alloy` [44], `DC` (The Duration Calculus) [58], `Event B` [1], `RSL` [39], `TLA+` [45], `VDM` [32, 33, 38], `Z` [57]

5.3 What Have I Otherwise Achieved ?

I think the following reasonably substantial elements can be said to have been achieved:

- *A comprehensive model of a substantial domain not described before.* It is all there ! Or at least – with much less effort than would otherwise be required – one can now see how to "complete" those context descriptions that are needed to express all those assumptions that need be stated when proving correctness of implementations.
- *Identifications of a number of "weak" spots in [30].*
  - The present Sect. 2.3.3 on Page 12 clarifies *mereology* issues of [30, Sect. 3.3]. So does Sect. 2.3.4 on Page 13.
  - The present Sect. 4.3 on Page 31 clarifies issues of *channel* analysis & description wrt. [30, Sects. 4.5.1, 4.5.2, 4.7.3].
- *Illustrating the importance of adhering strictly to the analysis & description principles laid down in [30] and to the order:*
  - first analysis & description of parts,
  - then analysis & description of unique identifiers,
  - then analysis & description of mereologies,
  - then compilation of parts,
  - then analysis & description of channels,
  - then analysis & description of behaviour signatures, finally analysis & description of behaviour definitions.

  Many times in this case study, more acting as an engineer, "cutting corners", I was lead astray – only to recover when realising that I was not following "strict procedures" !

  The "order" referred to above is also the order taken by the operational interpreter defined in [19, 22].

MORE TO COME

5.4 What Do I Plan to Have Done Next ?

I should like, myself or have others, to look at:

- *verification of the congruence of the operational, state machine definition of* transfer, *Items 85– 92 on Page 26 with respect to the planner behaviour, 126 – 126(d) on Page 37.*
- *validation* of this report,
- *identifier use/definition and type check,*
- *simplification of some behaviour definitions,*
- *reformulation of drone movement, i.e., the* `manoeuvre` *behaviour,*
- *clarification of the* `planner` *function's relation to [46],*

- *study of [50, 51, 52],*
- *study of [48],*
- *study of other literature,*
- *"completing" the term and the concept indexes,*
- etc.

5.5 Acknowledgements

I gratefully acknowledge:

- Dr Yang ShaoFa for having put me on to this subject, one that has entertained me now for a month (as this is written: 14. December 2017) – one that has allowed me to review and further improve, in some cases rectify, the method for manifest domain analysis & description first published in [30];
- Dr Klaus Havelund for referring me to Scott Smolka and his colleagues; that is to
- Dr Usama Mehmood who provided me with [46],
- To the 8 co-authors of Dr Mehmood; and to
- my wife for being patient with me during the last month (12.11–12.12.2017).

6 References

[1] Jean-Raymond Abrial. The B Book: Assigning Programs to Meanings *and* Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, England, 1996 and 2009.

[2] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.

[3] Dines Bjørner. Domain Models of "The Market" — in Preparation for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. Final draft versionhttp://www2.imm.dtu.dk/ db/themarket.pdf.

[4] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. Final versionhttp://www2.imm.dtu.dk/ db/ifac-dynamics.pdf.

[5] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [10, 13].

[6] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen. See [11, 14].

[7] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [12, 15].

[8] Dines Bjørner. A Container Line Industry Domain. Techn. report, Fredsvej 11, DK-2840 Holte, Denmark, June 2007. Extensive Drafthttp://www2.imm.dtu.dk/ db/container-paper.pdf.

[9] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.

[10] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press, 2008.

[11] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press, 2008.

[12] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press, 2008.

[13] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[14] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[15] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.

[16] Dines Bjørner. On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical University of Vienna, August–October 2010. http://www.imm.dtu.dk/~dibj/wfdftp.pdf.

[17] Dines Bjørner. Pipelines – a Domain Descriptionhttp://www.imm.dtu.dk/~ dibj/pipe-p.pdf. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[18] Dines Bjørner. Road Transportation – a Domain Description-http://www.imm.dtu.dk/~ dibj/road-p.pdf. Experimental Research Report 2013-4, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[19] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[20] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[21] Dines Bjørner. A Credit Card System: Uppsala Draft. Technical Report: Experimental Research, Fredsvej 11, DK–2840 Holte, Denmark, November 2016. http://www.imm.dtu.dk/~dibj/2016/credit/accs.pdf.

[22] Dines Bjørner. Domain Analysis and Description – Formal Models of Processes and Prompts. 2016. Extensive revision of [20]. http://www.imm.dtu.dk/~dibj/2016/process/process-p.pdf.

[23] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. 2016. Extensive revision of [9].

[24] Dines Bjørner. Weather Information Systems: Towards a Domain Description. Technical Report: Experimental Research, Fredsvej 11, DK–2840 Holte, Denmark, November 2016. http://www.imm.dtu.dk/~dibj/2016/wis/wis-p.pdf.

[25] Dines Bjørner. What are Documents? Research Note, July 2017. http://www.imm.dtu.-dk/~dibj/2017/docs/docs.pdf.

[26] Dines Bjørner. The Manifest Domain Analysis & Description Approach to Implicit and Explicit Semantics. *EPTCS: Electronic Proceedings in Theoretical Computer Science, Yasmine Ait-Majeur, Paul J. Gibson and Dominique Méry*, 2018. First International Workshop on Handling IMPlicit and EXplicit Knowledge in Formal Fystem Development, 17 November 2017, Xi'an, China.

[27] Dines Bjørner. To Every Manifest Domain a CSP Expression — A Rôle for Mereology in Computer Science. *Journal of Logical and Algebraic Methods in Programming*, (94):91–108, January 2018.

[28] Dines Bjørner. The Tokyo Stock Exchange Trading Rules. R&D Experiment, Fredsvej 11, DK-2840 Holte, Denmark, January and February, 2010. Version 1, 78 pages: many auxiliary appendiceshttp://www2.imm.dtu.dk/ db/todai/tse-1.pdf, Version 2, 23 pages: omits many appendices and corrects some errors.http://www2.imm.dtu.dk/ db/todai/tse-2.pdf.

[29] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. A JAIST Press Research Monograph # 4, 536 pages, March 2009.

[30] Dines Bjørner. Manifest Domains: Analysis & Description. *Formal Aspects of Computing*, 29(2):175–225, March 2017. DOI 10.1007/s00165-016-0385-z http://link.springer.com/-article/10.1007/s00165-016-0385-z.

[31] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. Extended versionhttp://www2.imm.dtu.dk/ db/pasadena-25.pdf.

[32] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978.

[33] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[34] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, 1980.

[35] Dines Bjørner. Urban Planning Processes. Research Note, July 2017. http://www.imm.dtu.-dk/~dibj/2017/up/urban-planning.pdf.

[36] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[37] G.B. Clemmensen and O. Oest. Formal specification and development of an Ada compiler – a VDM case study. In *Proc. 7th International Conf. on Software Engineering, 26.-29. March 1984, Orlando, Florida*, pages 430–440. IEEE, 1984.

[38] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[39] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[40] P. Haff and A.V. Olsen. Use of VDM within CCITT. In *VDM – A Formal Method at Work, eds. Dines Bjørner, Cliff B. Jones, Micheal Mac an Airchinnigh and Erich J. Neuhold*, pages 324–330. Springer, Lecture Notes in Computer Science, Vol. 252, March 1987. Proc. VDM-Europe Symposium 1987, Brussels, Belgium.

[41] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8), Aug. 1978.

[42] C.A.R. Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[43] C.A.R. Hoare. Communicating Sequential Processes. Published electronically: `http://www.usingcsp.com/cspbook.pdf`, 2004. Second edition of [42]. See also `http://www.usingcsp.com/`.

[44] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[45] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.

[46] Usama Mehmood, Radu Grosu, Ashish Tiwari, Nicola Paoletti, Shan Lin, Yang JunXing, Dung Phan, Scott D. Stoller, and Scott A. Smolka. *Declarative vs Rule-based Control for Flocking Dynamics*. In *Proceedings of ACM/SIGAPP Symposium on Applied Computing (SACC 2018)*. ACM Press, April 9–13, 2018. 8 pages.

[47] Ole N. Oest. VDM from research to practice (invited paper). In *IFIP Congress*, pages 527–534, 1986.

[48] Reza Olfati-Saber. Flocking for Multi-agent Dynamic Systems: Algorithms and Theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 13 March 2006. http://ieeexplore.ieee.org/document/1605401/; DOI: 10.1109/TAC.2005.864190; Thayer School of Engineering, Dartmouth College, Hanover, NH, USA.

[49] Martin Pěnička, Albena Kirilova Strupchanska, and Dines Bjørner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version-http://www2.imm.dtu.dk/ db/martin.pdf.

[50] Craig Reynolds. *Flocks, Herds and Schools: A Distributed Behavioral Model*. *SIGGRAPH Computer Graphics*, 21(4), August 1987. https://doi.org/10.1145/37402.37406.

[51] Craig Reynolds. *Steering Behaviors for Autonomous Characters*. In *Proceedings of Game Developers Conference*, pages 763–782, 1999.

[52] Craig Reynolds. OpenSteer, *Steering Behaviours for Autonomous Characters*, 2004. http://opensteer.sourceforge.net.

[53] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. Now available on the net: http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf.

[54] Douglas T. Ross. Toward foundations for the understanding of type. In *Proceedings of the 1976 conference on Data: Abstraction, definition and structure*, pages 63–65, New York, NY, USA, 1976. ACM. http://doi.acm.org/10.1145/800237.807120.

[55] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

[56] Albena Kirilova Strupchanska, Martin Pěnička, and Dines Bjørner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version-http://www2.imm.dtu.dk/ db/albena.pdf.

[57] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[58] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real–time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

A Indexes

A.1 Concept Definition Index

This index covers the general concepts of the domain analysis & description approach followed in this report. There are 20 concept definition references. I have only just begun indexing the concepts coverede in this report.

abstract type, 8
atomic part, 7

composite part, 8
concrete type, 8

discrete endurant, 7

endurant, 7
    discrete, 7
entity, 7

method fn. 15, 7
methodology fn. 15, 7

part, 7
    atomic, 7
    composite, 8
    sub-, 8

sort, 8
sub-part, 8

type, 8
    abstract, 8
    concrete, 8

unique identifier, 10

A.2 Term Definition Index

This index covers the specific terms of the universe of swarms of drones, etc., analysed & described in this report. There are 2 term definition references. I have only just begun indexing the terms of the special case stuy of this report.

geography, 8

universe of discourse, 8

A.3 Formula Indexes

The are 166 formula index references.

A.3.1 Sorts

Part Sorts
    AED 2(b), 8
    AOD 1(c), 8
    CA 12, 10
    CC 2(a), 8
    CM 10, 10
    CP 11, 10
    E 1(b), 8
    ED 3, 9
    G 1(a), 8
    OD 4, 9
    UoD 1, 8

Unique Ids.
    AEDI 13, 10
    AODI 13, 10
    CAI 16, 11
    CCI 13, 10
    CMI 14, 11
    CPI 15, 11
    EDI 17(a), 11
    EI 13, 10
    GI 13, 10
    ODI 17(b), 11

A.3.2 Types

 © D.Bjørner 2017. Fredsvej 11, 2840 Holte, Denmark. December 16, 2017: 08:21

Attribute Types
- BDIR = BI $\overrightarrow{m}$ SI-**set** 64, 21
- DDIR = DI $\overrightarrow{m}$ RoDD 66, 22
- DI = EDI |ODI 17, 11
- DYN = VEL × ACC × ORI × POS 52, 19
- EPS = P-**infset** 72, 24
- FDDIR = EDI $\overrightarrow{m}$ FP × FP* 69, 23
- ImG = A × L × W 55, 19
- ImG = A × L × W 57, 20
- MRoDD = RoDD 62, 21
- PFPL = FP* 54, 19
- SDIR = SI $\overrightarrow{m}$ DI-**set** 65, 21

Mereology Types
- CAM = EDI-**set** × CPI 29, 14
- CMM = DI-**set** × CPI 27, 14
- CPM = (CAI × CMI × GI) × Cmd 25, 13
- EDM = CMI×CAI×GI 31, 15
- GM = CPI×CMI×DI-**set** 34, 16
- ODM = CMI × GI 33, 16

Other Types
- $\mathbb{TI}$ Time Interval 36, 16
- $\mathbb{T}$ Time 35, 16
- A 70, 23
- ACC 50, 18
- AL 42, 17
- ALTI 48, 18
- BI, Business Identifier 63, 21
- CFP = EDI $\overrightarrow{m}$ FP 129, 39
- CuDD=(EDI $\overrightarrow{m}$ TiDYN)∪(ODI $\overrightarrow{m}$ TiDYN) 60, 20
- DIRECTION 49, 18
- DPOS 48, 18
- FP = FLE* 46, 17
- FPE = $\mathbb{T}$ × P 45, 17
- IncrDecrSPEEDperTimeUnit 50, 18
- L 73, 24
- LA 42, 17
- LATI 48, 18
- LO 42, 17
- LONG 48, 18
- ORI 51, 18
- P = LO×LA×AL 42, 17
- PITCH 51, 18
- RoDD=(EDI $\overrightarrow{m}$ TiDYN*)∪(ODI $\overrightarrow{m}$ TiDYN*) 60, 20
- ROLL 51, 18
- SPEED 49, 18
- TDIR = BDIR × SDIR × DDIR 67, 22
- TiDYN = $\mathbb{T}$ × POS ×DYN × ImG 58, 20
- VEL 49, 18
- W 74, 24
- YAW 51, 18

Part Types
- EDs = ED-**set** 3, 9
- ODs = OD-**set** 4, 9

A.3.3 Functions

Extract
- xtr_
  - D: AED → DI $\xrightarrow{\sim}$ ED 23, 12
  - D: AOD → DI $\xrightarrow{\sim}$ OD 24, 12
  - D: E → DI $\xrightarrow{\sim}$ ED 22, 12
  - D: UoD → DI $\xrightarrow{\sim}$ D 21, 12
  - dis: AED → DI-**set** 18, 11
  - dis: AOD → DI-**set** 19, 11
  - dis: UoD → DI-**set** 20, 11
  - Ds: AED → ED-**set** 7, 9
  - Ds: AOD → OD-**set** 8, 9
  - Ds: E → ED-**set** 6, 9
  - Ds: UoD → (ED|OD)-**set** 5, 9

Magics
- neighbours 44, 17
- time 37, 16

Observe Attributes: attr_
- BDIR: CP → BDIR 64, 21
- DDIR: CP → DDIR 66, 22
- DYN: ED → DYN 52, 19
- DYN: OD → DYN 56, 20
- EPS: G → EPS 72, 24
- FDDIR: CA → FDDIR 69, 23
- FFP: ED → FP 53, 19
- ImG: ED → ImG 55, 19
- ImG: OD → ImG 57, 20
- MRoDD: CM → MRoDD 62, 21
- PFPL: ED → PFPL 54, 19
- SDIR: CP → SDIR 65, 21

Observe Mereology: mereo_
- CA: CA → CAM 29, 14
- CM: CM → CMM 27, 14
- CP: CP → CPM 25, 13
- ED: ED → EDM 31, 15
- G: G → GM 34, 16
- OD: OD → ODM 33, 16

Observe Parts: obs_
- AED: E → AED 2(b), 8
- AOD: UoD → AOD 1(c), 8
- CA: CC → CA 12, 10
- CC: E → CC 2(a), 8
- CM: CC → CM 10, 10
- CP: CC → CP 11, 10
- E: UoD → E 1(b), 8
- EDs: AED → EDs 3, 9
- G: UoD → G 1(a), 8
- ODs: AOD → ODs 4, 9

Observe Unique Ids.: uid_
- AED: AED → AEI 13(c), 10
- AOD: AOD → AODI 13(d), 10
- CA: CA → CAI 16, 11
- CC: CC → CCI 13(e), 10

CM: CM → CMI 14, 11
CP: CP → CPI 15, 11
E: E → EI 13(b), 10
ED: ED → EDI 17(a), 11
G: G → GI 13(a), 10
OD: OD → ODI 17(b), 11

A.3.4 Channels

Channel Message Types
CA_ ED_ MSG = FP 113, 33
CM_ CP_ MSG = MRoDD 111, 33
CP_ CA_ MSG = EDI $\overrightarrow{m}$ FP 112, 33
D_ CM_ MSG = CuDD 110, 32
D_ G_ MSG = P 115, 34
G_ D_ MSG = ImG 115, 34

Declarations
attr_ DYN_ ch[edi]:DYN 119, 35
attr_ DYN_ ch[odi]:DYN 118, 35
attr_ L_ ch[g_i]:L 120, 35
attr_ W_ ch[g_i]:W 121, 35
ca_ ed_ ch[ca_i,edi]:CA_ ED_ MSG 113, 33
cm_ cp_ ch[cm_i,cp_i]:CM_ CP_ MSG 111, 33
cp_ ca_ ch[cp_i,ca_i]:CP_ CA_ MSG 112, 33
d_ cm_ ch[di,cm_i]:D_ CM_ MSG 110, 32
d_ g_ ch[di,g_i]:D_ G_ MSG 115, 34
g_ d_ ch[g_i,di]:G_ D_ MSG 115, 34

A.3.5 Behaviours

actuator:
CAI×(CPI×edis:EDI-**set**)→FDDIR→
in cp_ ca_ ch[cpi,cai]
out ca_ ed_ ch[cai,*] **Unit** 129, 39

compile
AED 94, 27
AOD 97, 28
CA 103, 29
CC 100, 28
CM 101, 29
CP 102, 29
ED 96, 28
EDs 95, 27
G 99, 28
OD 98, 28
UoD 93, 27
UoD 107, 30
UoD 106, 30

enterprise_ drone:
edi:EDI×(cmi:CMI×cai:CAI×gi:GI)→
((FPL×PFPL)×(DDYN×ALW))→
in attr_ DYN_ ch[edi],
g_ d_ ch[gi,edi],
ca_ ed_ ch[cai,edi]
out d_ cm_ ch[edi,cmi],
d_ g_ ch[edi,gi] **Unit** 139, 41

geography:
gi:GI×gm:(cpi:CPI×cmi:CMI×dis:DI-**set**)×...→
in cp_ g_ ch[cpi,gi],
d_ g_ ch[∗,gi]
out g_ cp_ ch[gi,cpi],
g_ d_ ch[gi,∗] **Unit** 152, 44

monitor:
cmi:CMI×(dis:DI-**set**×cpi:CPI)
→MRoDD→
in d_ cm_ ch[*,cmi]
out cm_ cp_ ch[cmi,cpi] **Unit** 122(a), 35

other_ drone:
odi:ODI×(cmi:CMI×gi:GI)×...→
in attr_ DYN_ ch[odi],
g_ d_ ch[gi,odi]
out d_ g_ ch[odi,gi],
d_ cm_ ch[odi,cmi]
Unit 132, 40

planner:
cpi:CPI×(cai:CAI×cmi:CMI×gi:GI)→
TDIR→
in cm_ cp_ ch[cmi,cpi],
g_ cp_ ch[gi,cpi]
out cp_ ca_ ch[cpi,cai] **Unit** 124(a), 36

A.3.6 Constants

ad_{is} 84, 25
ca_i 81, 25
cc_i 78, 25
cm_i 79, 25
cp_i 80, 25
e_i 75, 25
ed_{is} 83, 25
g_i 77, 25
o_i 76, 25
od_{is} 82, 25

B References to Swarms, Drones, UVAs, etc.

- Multi-hop Communications in a Swarm of UAVs
  Rachael Purta, Saurabh Nagrecha, Gregory Madey
  Department of Computer Science and Engineering,
  University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, Indiana 46556
  Email: rpurta@nd.edu, snagrech@nd.edu, gmadey@nd.edu

  Unmanned Aerial Vehicles (UAVs) are of increasing interest to researchers because of their diverse applications, such as military operations and search and rescue. The problem we have chosen to focus on is using a swarm of small, inexpensive UAVs to discover static targets in a search space. Though many different swarm models have been used for similar problems, our proposed model, the Icosystem Swarm Game, to our knowledge has not been evaluated for this particular problem of target search. Further, we propose to simulate the performance of this model in a semi-realistic communications environment. The challenge here is to find the optimal multi-hop configuration for the UAVs, so that they can find the most targets, avoid collision with each other as much as possible, and still communicate efficiently. We implement this through a weighted shortest-path problem using Dijkstras algorithm, with the weights being the transmission cost over distance. Testing has shown that our multi-hop communications perform, in terms of target-finding and collision avoidance with other UAVs, as well as an idealized communications environment.

- Towards A Swarm of Agile Micro Quadrotors
  Alex Kushleyev, Daniel Mellinger, Vijay Kumar
  GRASP Lab, University of Pennsylvania
  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.423.203rep=rep1type=pdf

  We describe a prototype 73 gram, 21 cm diameter micro quadrotor with onboard attitude estimation and control that operates autonomously with an external localization system. We argue that the reduction in size leads to agility and the ability to operate in tight formations and provide experimental arguments in support of this claim. The robot is shown to be capable of 1850^0/sec roll and pitch, performs a 360^o flip in 0.4 seconds and exhibits a lateral step response of 1 body length in 1 second. We describe the architecture and algorithms to coordinate a team of quadrotors, organize them into groups and fly through known three-dimensional environments. We provide experimental results for a team of 20 micro quadrotors.