Software Systems Engineering

From Domain Analysis via Requirements Capture to Software Architectures

Dines Bjørner
UNU/IIST,*P.O.Box 3052, Macau

Abstract

Based on an Air Traffic example we illustrate issues of Domain Analysis (of: air space, time tables, and traffic), of Requirements Capture (of: scheduling systems, and of air traffic control), and of Software Architectures for the above.

1 An overview of an example method

We explore a method for developing software for an air traffic system in which a Software Architecture is developed from a Requirements Capture, which is again developed from a Domain Analysis. We do not expect strict adherence to the temporal relation: Software Architecture from Requirements Capture and Requirements Capture from Domain Analysis — or: first Domain Analysis, then Requirements Capture and finally Software Architecture — but rather expect that once a software construction has been completed then its documentation follow the above suggested sequence.

The principles according to which the techniques are applied, are only claimed relevant wrt. the specific example. We have however also applied basically these principles to other infrastructure systems (railways, toll-ways, manufacturing industries, etc.).

The techniques and the tools used are those of the RAISE method, [20], respectively RSL, the RAISE Specification Language, [19]. RAISE is a comprehensive method for describing rather arbitrary software systems and for developing correct implementations from such descriptions. RSL allows descriptions that range from axiomatic, via applicative (functional) and imperative to concurrent, with these descriptions being embodied in modules: classes, objects and schemes. The RAISE/RSL proof system and implementation (refinement) relation allows you to "posit and verify" (construct and prove) correctness of stages of development, from abstract types to concrete types.

1.1 Application domain analysis

By 'domain' we loosely mean the application area, such a railways, air transport, manufacturing enterprise, health care, libraries, etc.

In this report we illustrate the domain of air traffic.

By a 'domain analysis' we understand an analysis, i.e. an identification of the system to be described together with the formal description of (parts) of the domain — possibly adorned with theorems (about the domain) and possibly their proof.

Each domain component is formally described and thus Domain Analysis creates theories of the domain — much like Maxwell's Equations form a theory of electro-magnetic wave propagation irrespective of any radio communication application.

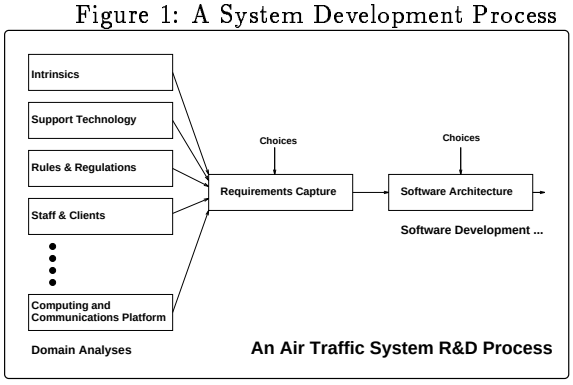
Domain Analysis is thus carried out without any reference to any software that might later be developed from requirements which in turn have been developed from a domain analysis.

It is important to observe, for the chosen application, that we decompose domain analysis into several parts: (i) intrinsics, (ii) support technology, (iii) [operator, resp. client] rules & regulations, (iv) operator, resp. client "behaviours", (v) domain economics, (vi) domain safety & dependability analysis, etc. We cover intrinsics, to some depth, and support technology and rules & regulations rather cursorily. See figure 1.
What the figure intends to express is that a development contains activities that produce documents for each of the boxes shown — not necessarily the order of their construction. What the "Process" figure does not show are the relations between the domain boxes. We shall try elucidate these:

- Intrinsics $\sqsubseteq$ Support Technology
- Intrinsics $\oplus$ Support Technology $\Rightarrow$ Plant
- (Plant || Regulations || Staff) $\Rightarrow$ Operations

*UNU/IIST: The United Nations' International Institute for Software Technology; Author's E-mail: db@iist.unu.edu; Fax: +853-712.940

Figure 1: A System Development Process



- Operations || Clients ⇒ Business
- Business ⋈ C&C Platforms ⇒ Efficiency
- Efficiency ⋈ Economics[1] ⇒ Profitability

The terms 'plant', 'operations', 'business', 'efficiency' and 'profitability' are initially non-technical terms. They are used (non-technically) by the customers who require the kind of computing systems we aim at. We use instead the terms: 'intrinsics', 'support technology', 'rules & regulations', 'staff', 'clients', 'platforms', and 'economics' — and then we postulate, through the above "formulas", that we can indeed address customer concerns.

It remains, however, to justify the above postulates ("meta-theory") through an appropriate theoretical study.

1.2 Requirements capture

By 'requirements' we loosely mean a set of statements that express desires for computerized support of some functions, operations and behaviours of an application domain.

By a Requirements Capture we mean a set of formal expressions of such statements — as well as the 'process' of capturing these specifications.

In this report, i.e. for the example chosen, we indicate the capture of requirements for (i) a traffic scheduling system, and (ii) an air traffic monitoring & control system.

1.3 Software architecture

By Software Architecture we loosely mean a document that describes a major aspect of a top-level structuring of the ensuing software that is: major component identification and interface design, together with the internal structuring of the major components and their basic implementation in terms of software mechanisms such as pipes, etc. This aspect is often called top level systems design.

[1] Among the "dotted" omitted domain analysis views (i.e. boxes) of the "Process" figure given earlier is that of Economics.

2 Intrinsics domain analysis

2.1 An Overview

By the 'intrinsics' (of a[n application] domain) we mean the bare essentials that fundamentally characterize the domain: that ideally singles the domain out from any other domain.

The choice as to what "belongs" to 'intrinsics' and what "belongs" to other sub-domains is, however, a pragmatic, hence subjective one.

After some experimentation the following "areas" were deemed exemplary part of the sub-domain of intrinsics:

Air-space: connections (relations) between airports, air-domes, and airways; and the operations of putting new airports, air-domes, and airways into service, modifying existing ones, or removing airports (hence air-domes) or airways;

Time tables: which flights fly where and at which arrival and departure times; and the operations of time tabling: creating new time tables from scratch, modifying existing ones (incl. adding new, changing present, or removing present entries);

Traffic: the positions, at any time, of any flight; and the operations of scheduling and re-scheduling, dispatching, redirecting and cancelling flights.

The following "areas" were deemed "outside" (i.e. not part of) intrinsics: (a) the (here: support) technology that enters into the operation of air traffic; (b) the civil (and other, for example military) aviation rules & regulations; (c) staff (i.e. air traffic controller, pilot, etc.) behaviours, etc. These latter will here be treated as separate sub-domains. In doing so we will observe that we refer to concepts and facilities (i.e. [state] components, functions, operations and behaviours) of other sub-domains, notably the intrinsics. Hence we begin by presenting that which is common to, i.e. shared by most other sub-domains.

2.2 Some Illustrations

Before we embark on a detailed analysis, let us give some hints, in the form of concrete type models and example expressions of the modeled types.

An air space example "picture" See figure 2.

Figure 2: An Air Space "Picture"

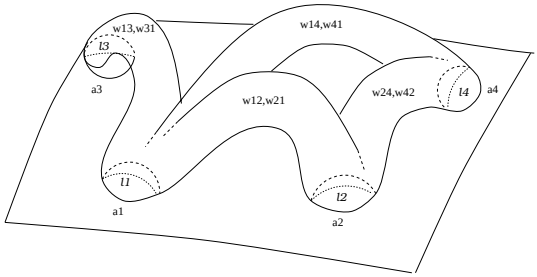
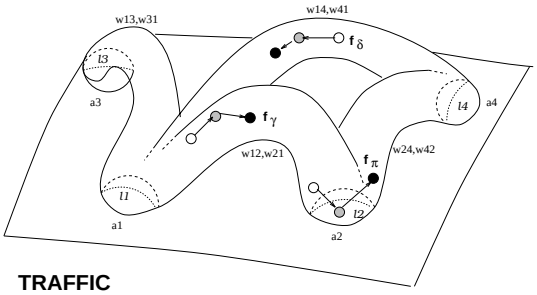


Figure 3: An Air Traffic "Picture"



An air space model and an example:

```
type G = A ~m~> (L × (A ~m~> W))
value
    g: [ a1 ↦ ( l1 , [ a2 ↦ w12,
                       a3 ↦ w13,
                       a4 ↦ w14 ] ),
         a2 ↦ ( l2 , [ a1 ↦ w21,
                       a4 ↦ w24 ] ),
         a3 ↦ ( l3 , [ a1 ↦ w31 ] ),
         a4 ↦ ( l4 , [ a1 ↦ w41,
                       a2 → w42 ] ) ]
```

A time table model and an example:

```
type TT = F ~m~> (A ~m~> (T × T))

value
  tt: [ fα ↦
            [ a1 ↦ (t3,t4),
              a2 ↦ (t9,t11) ],
        fβ ↦
            [ a3 ↦ (t1,t2),
              a1 ↦ (t5,t7) ],
        fγ ↦
            [ a1 ↦ (t13,t14),
              a3 ↦ (t16,t18) ],
        fδ ↦
            [ a1 ↦ (t6,t8),
              a4 ↦ (t19,t20) ],
        fπ ↦
            [ a2 ↦ (t10,t12),
              a4 ↦ (t15,t17) ,
              a1 ↦ (t21,t22) ] ]
```

An air traffic "picture" . See figure 3.

An air traffic model and an example:

```
type
    TF = T ~m~> (F ~m~> E)
    TR = T ~m~> (T × E)*
value
    cTFtoTR: TF ~→ TR
```

⟨ ..., (t,[fγ ↦ eγ, fδ ↦ eδ, fπ ↦ eπ]),
(t′,[fγ ↦ eγ1, fδ ↦ eδ1, fπ ↦ eπ1]),
(t″,[fγ ↦ eγ2, fδ ↦ eδ2]), ... ⟩

fγ: ⟨ .. (t,eγ),(t′,eγ1),(t″,eγ2), .. ⟩

2.3 Air spaces

We will now begin our series of description experiments. First noting down concrete types, later abstract types, and finally observer functions and generator operations over these. Initially we give their signatures. Eventually we shall express a number of axioms that relate observer functions and generator operations.

2.3.1 Concrete types

Airports have names and are otherwise here considered to be further undefined entities.

type A

With airports we associate air-domes: a continuous space, or a composition of connected continuous spaces, that "surround" the airport: which denotes its geographical, for example three dimensional (x, y, z) or polar coordinate, spatial extension. In the air traffic control terminology air-dome[2] is often referred to as terminal area (TMA), a controlled area, or a control zone, usually smaller than control area.

type D

Section 2.3.2 elaborates on D.

An air space[3] is a composition of airports, their air-domes and the airways between (some, but not

[2] We hyphenate 'air-dome' to indicate that it is a constructed term.

[3] We write 'air space' to distinguish from 'airspace': the air space over a country considered as a property of that country.

necessarily all) airports. An airway is a continuous space between a pair of airports and includes their air-domes.

type
W
S'' = A $\overrightarrow{m}$ (D × (A $\overrightarrow{m}$ W))

Section 2.3.2 elaborates on W.

The model just given defines an infinite set of (simplified) air spaces.

A well-formed subset of these have all airports that are used also being defined, that is: there are no dangling airways (i.e. edges in a graph which have airport names as node labels and airways as further undefined edges). Further constraints are mentioned next.

We speak now of internal and of external airports. An internal airport is one whose air-dome is defined. An external airport is one whose air-dome is not defined. From an internal airport there may be zero, one or more airways leading to (or coming from) other internal airports. Such airways include the air-domes of the pairs of connected airports as proper "ends". We may think of the two "ends" of an airway to be "anchors". From an internal airport there may additionally be zero, one or more airways leading to [or coming from] external (output[, respectively input]) airports. Corresponding output, respectively input airways need not have their airways "anchored" in both airports' air-domes, but definitely in the air-dome of the internal airport. Some airways may connect a pair of external airports: such airways can be said to provide fly-over possibilities. Finally: all domes and airways are contained within an air space volume (v:V).

type
S' = ias:A-**set**
× g:(A $\overrightarrow{m}$ (ad:AD × aws:(A $\overrightarrow{m}$ W)))
× oas:A-**set**
× sp:V
AD = undef | d(D)
V

As mentioned above, for S'', internal, used airports must be defined, i.e. have their air-domes defined.

2.3.2 A theory of 3-dimensional bodies

So far two three-dimensional (i.e. spatial) bodies have been introduced: D, W, V. We now introduce a fourth spatial entity: a surface — so that we can identify where airways are "glued" to, or intersects, space spheres (volumes).

type
D, W, V = B
Sur

We have spoken about an airway being a spatial body being "anchored" by two (internal airport) air-domes or by one internal, either the target airport's air-dome if "the other" (first) airport is an input airport, or the source airport's air-dome if "the other" (last) airport is an output airport:

value
Two_Domes: D × W × D → **Bool**
Fst_Dome: D × W → **Bool**
Lst_Dome: W × D → **Bool**
Domes: W → D × D
axiom
forall d,d′:D, w:W •
Two_Domes(d,w,d′) ⇒
Fst_Dome(w,d′) ∧ Lst_Dome(d,w)
Domes(w) = (d,d′) == Two_Domes(d,w,d′)

In contrast we could speak of an air-dome being contained within, i.e. being a proper subset of an airway, and two distinct air-domes being disjoint:

value
Subset: B × B → **Bool**
Intersect: B × B → **Bool**
axiom
forall d,d′:D, w:W •
d ≠ d′ ⇒ ∼Intersect(d,d′) ∧
Two_Domes(d,w,d′) ⇒
Subset(d,w) ∧ Subset(d′,w)

Airways are allowed to intersect, air-domes not.

Given an airway we can speak of its length, or rather the minimum and maximum length of an airway:

type
LI = L × L
L = Rat
value
W_Length: W → LI

For more on intervals (of lengths) we refer to the next Section 2.3.3.

One may speak of an airway being a "reasonably smooth" connection or corridor:

value Smooth: W → **Bool**

We may impose a constraint:

axiom forall w:W • Smooth(w)

An airway may be glued to an air space volume; and glue points form surfaces: zero, one or two!

```
value
  is_Glued: W × V → Bool
  Glues: W × V → Sur-set
axiom
  forall w:W, v:V • 0 ≤ card Glues(w,v) ≤ 2
```

2.3.3 Towards a theory of intervals — I

Intervals, say in the form of pairs of rational numbers:

```
type
  I = {| (b,e) | b,e:Rat • b<e |}
  TI = T × T = I
  LI = L × L = I
  L,T = Rat
```

occur in different contexts. Three obvious classes of intervals are:

- the interval, ti:TI, of time an aircraft is at a gate or in an airport, as from the arrival to the departure times,
- the interval, ti:TI, of time that an aircraft is in the air, that is: the flying between two airports, as from the departure time from one airport to the arrival time at another airport,
- the interval, ti:TI, of minimum to maximum flying time, and
- the interval, li:LI, of minimum to maximum distance, or length, within an airway, that is: between two airports.

The first three items above are further commented upon in Section 2.4.1.

2.3.4 Abstract types

In section 2.3.1 we gave a concrete type model for S, the class of air spaces. In this section we give a description of the air spaces without giving a model of their components. We thus restrict ourselves to present sorts and a number of functions which extract, from Cartesian sorts, other sorts, such as AD, D, and W, or simple sets or Cartesian products:

```
type
  S, A, AD, D, W, V
value
  is_Empty_S: S → Bool
  Obs_As: S → A-set
  Obs_iAS: S → A-set
  is_Internal_A: A × S → Bool
  Obs_xAs: S → A-set
  is_External−A: A × S → Bool
  Obs_inAs: S → A-set
  is_Input−A: A × S → Bool
  Abs−OuAs: S → A-set
  is_Output_A: A × S → Bool
  Obs_AD: A × S ~→ AD
  Obs_dAs: A × S → A-set
  is_Conn_AA: (A × A) × S → Bool
  Obs_W: (A × A) × S ~→ W
```

Partiality of some functions require the definition of suitable pre-conditions.

```
value
  Obs_dAs(a): pre is_Internal_A(a)
  Obs_W(a,a′): pre is_Conn_AA(a,a′)
```

2.3.5 Operations on air spaces

In real life new airports are brought into service, old airports are taken out of service, and existing airports are temporarily closed (and subsequently re-opened) or are modified — by some modification m:M, which we leave undefined.

In real life the scope of airports that we consider is changed: external airports become internal or "disappear", solely input airports also become output airports, and vice versa.

In real life airways are changed, closed, re-open, etc.

We present the signature of these state-changing operations, but first a pre-amble: CON is the type of state to state changing functions.

```
type
  CON = S ~→ S
```

Now the operation signatures:

```
value
  New_S: S
  New_A: A → CON
  Old_A: A → CON
  Cls_A: A → CON
  Opn_A: A → CON
  Mod_A: A × M → CON
  mk_In_A: A → CON
  rm_In_A: A → CON
  mk_Out_A: A ~→ CON
  chg_W: (A × A) × W ~→ CON
  cls_W: (A × A) ~→ CON
  opn_W: (A × A) ~→ CON
```

Suitable definitions can now be given of these operations — some require the definition of an appropriate pre condition. For example:

```
value
  chg_W(a,a′,w)(s): pre {a,a′} ⊆ Obs_As(s) ∧ is_Conn_AA(a,a′)
```

2.4 Time tables

2.4.1 Concrete types

We can look at time tables in several ways:

```
type
  TTa = A →m (F →m (T × T))
  TTf = F →m (A →m (T × T))
```

The first model emphasizes flights in and out of airports, the latter emphasizes flights.

Time tables must be well-formed:

```
wf_TTa: TTa → Bool
wf_TTf: TTf → Bool
```

Let ttf, f, f′, a, a′, a″ be related as follows:

```
ttf:TTf, f,f′:F, a,a′,a″:A
f,f′ ∈ dom ttf,
      f ≠ f′
a,a′ ∈ dom ttf(f),
      a ≠ a′
      a,a″ ∈ dom ttf(f′),
```

Well-formedness conditions for any given time table may include:

1. For any flight arrival/departure intervals (at distinct airports) do not overlap.

   ```
   let ((b,e),(b′,e′)) = ((ttf(f))(a),(ttf(f))(a′)) in
   e<b′ ∨ e′<b
   end
   ```

2. For any airport, pairs of arrival times, respectively pairs of departure times for distinct flights have a minimum time separation — that is no congestion.

   ```
   let ((b,e),(b′,e′)) = ((ttf(f))(a),(ttf(f′))(a)) in
   b−e > min_sep ∧
   b′−e′ > min_sep ∧
   e<b′ ⇒ b′−e > min_sep ∧
   e′<b ⇒ b−e′ > min_sep
   end
   ```

   This constraint can be refined: for any one runway of any one airport a suitable rewording of the above should hold. To tackle such a refinement we need further refine the notion of an airport. An airport could for example have several concurrently operable runways. Correspondingly two or more A's may in actuality designate different runways of the same airport — with corresponding air-domes now overlapping, or being identical!

3. Some airports may not allow scheduled arrivals or departures during certain time intervals, for example during night hours.

The two kinds of time tables, TTa and TTf, "commute":

```
value
  TTf_to_TTa: TTf → TTa
  TTa_to_TTf: TTa → TTf
axiom
  forall ttf:TTf, tta:TTa •
      wf−TTf(ttf) ⇒
        TTa_to_TTf(TTf_to_TTa(ttf)) = ttf ∧
      wf−TTa(tta) ⇒
        TTf_to_TTa(TTa_to_TTf(tta)) = tta
```

The above time tables, let us refer to them as absolute time tables, are not seen as "modulo-something". That is: we can think of another form of time table, let us refer to them as relative time tables, which is modulo the season and the day of the week! An absolute time table can be thought of as having been expanded from a relative time table. Thus a flight identity in a global time table can be thought of as [further] adorned with day/date information!

2.4.2 Invariant: Air space & time table

Clearly we can only have non-stop flights between existing and directly airway connected airports; and flying times should be commensurate with airway (corridor) lengths:

```
inv_S_TT: S × TT →~ Bool
  /* Only flights between airway connected airports */
  /* and with airway length commensurate flying times */
```

2.4.3 Abstract types

We define the sort TT of time tables, the sort R of routes, the sort J of journies (where a journey is a sequence of routes [and perhaps some more]), and a number of observation functions:

type
TT, R, J
value
Obs_Fs: TT → F-**set**
Obs_J: TT × F $\xrightarrow{\sim}$ J
Obs_dTaT: TT × F × A $\xrightarrow{\sim}$ T × T

2.4.4 Operations on time tables

One may add new, delete old and modify existing time table entries:

value
Add_TT: (F × J) × TT $\xrightarrow{\sim}$ TT
Del_TT: F × TT $\xrightarrow{\sim}$ TT
Mod_TT: F × (A × (T × T)) × TT $\xrightarrow{\sim}$ TT

2.5 Air traffic

2.5.1 Concrete types

By a flight we understand the trajectory formed by the position of an aircraft, designated by some f:F, over a continuous time interval: from a (dense) set of times when the aircraft is on the ground, via a (dense) set of times when the aircraft is taking off, in the air ('in flight') and landing, to a (dense) set of times when the aircraft is again on the ground. For each time the trajectory gives the position of the aircraft. A position, p:P, is a 3-dimensional body, some "envelope" around the aircraft, its "sphere" of integrity.

Traffic is now the composite of all flight trajectories: a function from time to functions from flight designators to positions.

type
TF = T $\xrightarrow{\sim}$ (F $\xrightarrow[m]{}$ P)
TFs = T $\xrightarrow{\sim}$ (F $\xrightarrow[m]{}$ P × Sch_Info)
TFr = T $\xrightarrow{\sim}$ (F $\xrightarrow[m]{}$ P × Real_Time_Info)
Tra = P^{ω}
Trajectory = (T $\xrightarrow{\sim}$ P)
value
Tra_f_Trajectory: Trajectory $\xrightarrow{\sim}$ Tra

TFs stands for scheduled flights: those that are planned and possibly ongoing (i.e. in flight); TFr stands for real flights; and TF for an abstraction of either of these where we have suppressed auxiliary information such as schedule information or real time information. Tra stands for infinite length sequences of (trajectory) positions.

A traffic may very well contain trajectories that "touches airport ground" three or more times: initial and final airport plus zero, one or stop-over airports. We can only determine whether a trajectory "is on an airport ground" if we are also given an air space, s:S.

Well-formedness Conditions:

- Continuous flights: No holes

  We also require that if a flight, f, is recorded (in either tf:TFs, tf:TFr or tf:TF) at times t and t' to be in flight, i.e. t and t' are in the domain of tf for flight f then at all times between t and t' that f is recorded in TF (etc.).

- Smooth trajectories

  The trajectory formed by all flight positions between such times is continuous. A trajectory is necessarily an infinite sequence of positions.

One may think that the next constraint is intrinsic:

- No collisions

  For a traffic, tf:TF, to be well-formed we require that no two positions of different flights at any time overlap — such would amount to a crash (or collision), whether in the air or on the ground.

But since accidents do occur we do not include it as an intrinsic property. Instead we define suitable collision or near-miss observation functions:

value
Crash: TF $\xrightarrow{\sim}$ (T $\xrightarrow[m]{}$ F-**set**)
Near−Miss: TF $\xrightarrow{\sim}$ (T $\xrightarrow[m]{}$ F-**set**)
Continuous_Flights: TF $\xrightarrow{\sim}$ **Bool**
Smooth_Flights: TF $\xrightarrow{\sim}$ **Bool**
is_wf_TF: TF → **Bool**
is_wf_TF(tf) ≡
  Continuous_Flights(tf) ∧
  Smooth_Flights(tf)

The Crash function observes, for any time in tf:TF, the set of all the flights that are crashing, i.e. whose positions interfere with one another. Near-Miss similarly observes near misses, i.e. pairs or more of flights which are not (yet) crashing but whose distance is less than the minimum safe distance.

2.5.2 Invariant: Air space & traffic

Given an air space, s:S, and a traffic we can check whether the trajectories are within the airways: Initial and final trajectory positions are those of air-domes; intermediate airport positions are "reasonable"; and trajectories fall within airways:

value inv_S_TF: S × TF → **Bool**

2.5.3 System invariant

Given all three major components: an air space, s:S, a time table, tt:TT, and a traffic, tf:TF, we can check whether flights are on time: Flights start on time, flights land on time, and real flights must be in time table:

value inv_S_TT_TF: S × TT × TF → **Bool**

We are not checking whether all flights recorded in the time table are indeed flying, but we are checking that actual, recorded, real time flights are indeed time tabled.

2.5.4 Invariant: scheduled and real traffics

Given a scheduled and a real, observed, traffic, the latter usually up to a time less that or equal to the largest time recorded in a traffic schedule, we can check whether a real (observed) traffic follows a scheduled traffic:

value inv_TFs_TFr: TFs × TFr → **Bool**

Given a scheduled and a real ,(observed) traffic one can observe a first time, if any, where a real flight is not on course with respect to its schedule:

value
is_TF_Disruption: TFs × TFr $\tilde{\to}$ **Bool**
TF_Disruption: TFs × TFr $\tilde{\to}$ (T × F)

2.5.5 Abstract types

We introduce the sorts of traffic, positions and trajectories:

type TF, P, Tra, Trajectory

We assume the functions that observe the first and last times of TT and the domain of TT as well as whether such a domain is dense.

We introduce the observation functions:

value
Obs_Fs: TF → **F-set**
Obs_bTeT: F × TF $\tilde{\to}$ T × T
Obs_Tra: F × TF $\tilde{\to}$ Tra

In a trajectory we can speak of the first and last position and the infinite sequence of positions, but we cannot compute the last position:

value
Obs_bP: Tra → P
Obs_eP: Tra → P

2.5.6 Operations on air traffic

We can add a flight (with its trajectory) to a traffic, we can cancel a flight, and we can modify a flight by changing its trajectory:

value
New_F: F × (T $\tilde{\to}$ P) × TF $\tilde{\to}$ TF
Cancel_F: F × TF $\tilde{\to}$ TF
Modify_F: F × (T $\tilde{\to}$ P) × TF $\tilde{\to}$ TF

The operations are partial since we have to ensure that input arguments are well-formed (smooth, continuous), and that resulting traffics are likewise.

3 Requirements capture

We consider the following groups of concerns as entering the Requirements Capture process and thus to be covered by a requirements specification.

Requirements capture include separate considerations of:

Refinement of domain specificities: We often find that one never "completes" a domain analysis: that further issues that ought be treated during Domain Analysis crop up, ie. are inspired by issues otherwise dealt with in Requirements Capture.

Computational behavioural characteristics: Is the computing system to be implemented characterized by its: *distributability, concurrency, reactiveness (interaction), clocks (local or global), dependency, safety criticality, performance,* etc.?

Answers to, i.e. decisions on the above usually leads to refinements of the Domain Analysis descriptions of previous stages, or to new such domain descriptions being added to the set already researched and developed.

We will not illustrate this point in the current report but refer to [17] which investigates techniques for developing a domain analysis that seemingly portrays a 'global state system' into one reflecting a 'distributed state system'.

Computing systems boundary: The Domain Analysis covered many "laws" about air space, time tables, and air traffic. Some of these laws are now to be understood as assumptions to be relied upon during Requirements Capture, Software Architecture and the further software development.

Computer & communications system platforms: Here we are concerned with the hardware and software that is envisaged for the implementation: main-frame, client-server (more specifically: *MS/DOS, NT/Windows, Windows'95, UNIX,*

VAX/VMS, IBM PC-compatible, X11-Windows w./ Widgets, Novell Netware, etc.

Decisions on the above leads us to further analyze these computer and communications platform components, including presenting formal descriptions of these as part of the Domain Analysis.

We will not illustrate this point in the current report.

General domain requirements: We will focus only on this aspect in this report and will do so in section 3.2.

3.1 Computing systems boundary

To understand the issues involved under the heading of 'computing system boundary' we bring the Air traffic control requirements example:

The system boundary excludes the air traffic: it can only be observed: "sensed" through support technology such as radar etc. We cannot control it directly, only hope that messages sent from air traffic controllers to pilots (etc.) concerning possible flight course (trajectory) adjustments issued so as to avoid for example collisions are followed. We cannot insure that they are indeed followed. So an air traffic control system receives and sends messages: it receives sensed data on for example flight positions and radio communicated conversations with pilots, etc. The same air traffic control system sends messages to pilots, to support technology (like radar) that activates re-sensing etc.

So major parts of the system modeled during Domain Analysis— is "outside" the computing system to be implemented. This has implications both for the mapping of external to internal states, their input and ongoing update, cf. section 3.3.1, as well as for the way in which the basic functions and operations behave, cd. section 3.3.2.

More specifically: the requirements shall decide upon the set of interfaces: those afforded by support technology sensors and activators and those afforded by staff and client message links.

3.2 General domain requirements

Requirements brings together and further enrich (i.e. algebraically extend, mo-del-theoretically further refine) issues of the combined domains.

In this Subsection we make an attempt to enumerate all the issues that requirements capture must cover wrt. the domain analyses. A subsequent Subsection (3.3) then illustrates these general issues wrt. the various requirements for different air traffic domain software packages (sub-systems).

1: Real States: External to internal state requirements: to what extent must the internal state of the software system reflect the actual state of the system.

2: State Input: Requirements on the initial input of the external state: Editing, vetting, validation, experimental computation, etc.

3: State Updates: Requirements on recording changes to external state, incl. frequency of (reason for) sampling.

4: Commands: Base Functions and their Invocation: Which functions, operations and behaviours of the application domain are to be supported and how: semi- or fully automatically.

5: Safety Criticality: Identification of which failures will be handled, and how. For example: reconciliation of apparent discrepancies between component domain models.

6: Dependability: Identification (estimation) of failure rates (probabilities), computation (estimation) of required dependability — relative to each individual failure source and to some combined failure sources.

7: Auxiliary Functions and their Invocation: Desired software which offers above features usually allow additional monitoring (statistics, and sometimes control) functions.

8: CHI Facilities: Facilities for displaying state information and results of computations: 3D and virtual reality animation, etc.

3.3 Specific requirements

In order to see the full consequences of having first established careful (formal) descriptions of the application domain we illustrate the capture of requirements for two different sub-systems.

Scheduling and Re-scheduling System: Given an air space and a commensurate time table, scheduling means: constructing a "best" well-formed scheduled traffic invariant with both the air space and the time table.

Re-scheduling is invoked whenever current traffic deviates from (a previous) scheduled traffic. (We may say that the scheduled traffic has been disrupted.) Re-scheduling then means: constructing a "best" alternative new scheduled traffic, as above, such that the new scheduled traffic up till the time deviation has been observed coincides with the previous schedule and such

that the new schedule as from that time on brings traffic back to "normal" (i.e. follows the original time table) "as soon as possible". Usually re-scheduling also means introducing a temporary time table for the period between the time disruption has been observed till "traffic is restored", that is: till the current traffic (at that time) coincides with the originally (previously) scheduled traffic.

Air Traffic Control System: Given an air space and scheduled and real, i.e. current traffic, air traffic control means: to dispatch flights (i.e. to monitor and guide departing flights, that is: take-off) out of air domes, to monitor and guide flights along airways, and to monitor and guide approaching (arriving, that is: landing) flights. Air traffic control determines whether flights are deviating from schedule and, if so, what form of temporary re-scheduling must be enacted.

For some of the 8 general domain requirements issues we will now relate these to each of the two application alternatives.

The reason why we treat two different application (sub-)areas is to show how they all re-use significant parts of the intrinsics domain analysis.

The message therefore is this:

In capturing requirements for any application within the larger domain, it is most likely very useful to have done Domain Analysis of a larger subset of the application domain that that which is immediately relevant to the specific computerization being requirements captured. And: Domain Analysis can then be re-used over a larger subset of specific computing supports.

We are not saying, however, that we must complete all of Domain Analysis — not even all of its intrinsics — for "the entire" applications domain, before we can tackle the requirements for any one software sub-system, We find that we cannot always

In the following six subsections we sketch varieties of requirements.

3.3.1 State mapping

Real States:

1. Scheduling System:

   State components: Air space and time table: S, TTf

   Re-Scheduling System:

   State components: Air space, time table, scheduled and current flight positions: S, TTf, TFs, TFr

2. Air Traffic Control System:

   State components: Air space, time table, scheduled and real traffic: S, TTf, TFs, TFr

From the above we observe how different sub-applications, usually developed in isolation from one another, require overlapping subsets of the "grand" external state to be represented internally. To safeguard, therefore, against future mismatches between such different sub-application software packages' implementation of such state components, it is therefore advisable that a careful analysis first be made wrt. computable representations of as encompassing an external grand state as foreseeable before any requirement decisions are made wrt. the specific packages' state "consumption". The domain analyses serves to make such an assessment as consistent and complete as possible irrespective of any application.

Initial State Input:

1. Scheduling & Re-Scheduling System:

   States input and input functions: Air space and time tables; air space and time table well-formedness and invariants: S, TTF; is-wf-S, is-wf-TTf, inv-S-TTf

2. Air Traffic Control System:

   States input and input functions: Air space, time tables, scheduled and real air traffic; S, TTf, TFs, TFr well-formedness: is-wf-S, is-wf-TTf, is-wf-TFs, is-wf-TFr invariants: inv-S-TTf, inv-S-TFs, inv-S-TTf-TFs, inv-S-TTf-TFr

We consider well-formedness and invariant checks, as well as prior data vetting to be indispensable parts of initial state input, as well as state update.

Initial[izing] input is thus seen as a major component of a resulting software system. Often this fact is initially overlooked with the consequence that resulting software "patches" rather ad hoc input systems onto a therefore ill thought out software architecture. And often these patches occur late in the development of the software giving rise to inconsistent treatment of states.

State Updates:

1. Scheduling & Re-Scheduling System:

   States updated, sampling frequency: Air space and time table changes, phasing old schedules out and new [re-schedules] in.

   Issues: Hard real-time: for re-scheduling maybe of the order of 5–10 minutes.

2. Air Traffic Control System:

   States updated, sampling frequency: Air space, time table, scheduled and real air traffic.

   Issues: Hard real-time: definitely so; order seconds (10-60 ?).

The question of how often various state values are to be sampled, and whether in response to interrupts or through polling is a crucial one. Classical laws of control theory must be observed in order to ensure Liapunow stability on one hand and timeliness (validity) of data on the other hand.

3.3.2 Function, operation and behaviour requirements: Sketches

1. Scheduling & Re-Scheduling System:

   Initial Scheduling vs. Re-scheduling functions.

   Initial Scheduling experiments (simulations).

   Re-scheduling: disruption notification etc.

   Invocation: yearly to seasonal for scheduling, "on-line, real-time" for re-scheduling.:

2. Air Traffic Control System:

   Aircraft dispatch control, departure and approach monitoring and advice, disaster prevention, disaster monitoring & control, etc.

   Invocation: constant polling and readiness for interrupts/exceptions.:

4 References

Basic references to air traffic systems are [22, 15, 18, 29, 21, 30]

5 Summary and Conclusion

5.1 Summary

5.2 Acknowledgements

References

[1] Gregory Abowd, Robert Allen, and David Garlan. Using Style to Understand Descriptions of Software Architecture. In *Symposium on Foundations of Software Engineering*, Redondo Beach, CA, USA, December 1993. SIGSOFT'93.

[2] Civil Aviation Authority. *Scottish & Oceanic Air Traffic Control Centres*, volume Doc. No. 259 of *National Air Traffic Services*. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 19??

[3] Civil Aviation Authority. *Air Traffic Management in the United Kingdom: Memorandum by the Civil Aviation Authority to the House of Commons Transport Committee*, volume CAP 537. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1988. ISBN 0 86039 342 9, GBP 4.

[4] Civil Aviation Authority. *Air Traffic Management in the United Kingdom: Second Memorandum by the Civil Aviation Authority to the House of Commons Transport Committee*, volume CAP 540. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1988. ISBN 0 86039 351 8, GBP 1.50.

[5] Civil Aviation Authority. *Strategies for making good use of airspace 1989-1995: Advice to the Secretary of State for Transport*, volume CAP 546. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1988. ISBN 0 86039 366 6, GBP 4.

[6] Civil Aviation Authority. CCF — Handling London's Air Traffic in the nineties, 1990.

[7] Civil Aviation Authority. *Controlling Britain's Air Traffic*, volume Doc. No. 420 of *National Air Traffic Services*. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1990.

[8] Civil Aviation Authority. *Outline of a method for the Determination of Separation Standards for future Air Traffic Systems*, volume CAP 91010. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1991.

[9] Civil Aviation Authority. *ATC Examination Syllabus: Section A: Aviation Law and Air Traffic Control Procedures*, volume CAP 390. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1992. ISBN 0 86039 530 8, GBP 2.00.

[10] Civil Aviation Authority. *ATC Examination Syllabus: Section D: The Approach Control Rating*, volume CAP 390. Civil Aviation Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1992. ISBN 0 86039 479 2, GBP 2.00.

[11] Civil Aviation Authority. *ATC Examination Syllabus: Section E: Approach Radar Control and Radar Theory*, volume CAP 390. Civil Aviation

Authority London, Greville House, 37 Gratton Road, Cheltenham, England, 1992. ISBN 0 86039 506 5, GBP 2.00.

[12] Civil Aviation Authority. Scottish and Oceanic Air Traffic Control Centre - an introducation, 1993.

[13] Civil Aviation Authority. Development Plan For the UK Air Traffic Control system, 1994. ISBN 0 86039 591 X, GBP 7.5.

[14] Air Traffic Services Standards Department. *Approval of Air Traffic Control Units.* Civil Aviation Authority, Greville House, 37 Gratton Road, Cheltenham, England, August 1991. ISBN 0 86039 488 3, GBP 15.

[15] Graham Duke. *Air Traffic Control.* Ian Allan Publishing, Terminal House, Station Approach, Sheperton, Surrey TW17 8AS, 5th ed edition, 1994.

[16] David Garlan. Formal Approaches to Software Architecture. In *Workshop on Studies of Software Design*, Heidelberg, Germany, May 1994. Springer-Varlag.

[17] Chris W. George. A Theory of Distributing Train Rescheduling. Technical report, UNU/IIST, P.O.Box 3058, Macau, April–August 1995.

[18] Dave Graves. *A Layman's Guide to United Kingdom Air Traffic Control.* Airlife Publishing Ltd., 101 Longden Road, Shrewsbury SY3 9EB, England, 2nd ed edition, 1993. ISBN 1 85310 407 8.

[19] The RAISE Language Group. *The RAISE Specification Language.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1195.

[20] The RAISE Method Group. *The RAISE Method.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[21] Nancy G. Leveson, Mats Per Erik Heimdahl, and Holly Hildreth. Requirements Specification for Process-Control Systems. *Transactions on Software Engineering*, 20(9):684–707, September 1994. **Note:** Illustrates requirements for an industrial aircraft collision avoidance system [TCAS II].

[22] Michael S. Nolan. *Fundamentals of air traffic control.* International Thomson Publishing, Wadworth Publishing Company, Belmont, California 94002, 2nd edition, 1994. ISBN 0-534-23058-X.

[23] International Civil Aviation Organization. Units of Measurement To Be Used In Air and Ground Operations, July 1979.

[24] International Civil Aviation Organization. Supplement To Units of Measurement To Be Used In Air and Ground Operations, May 1988.

[25] International Civil Aviation Organization. Supplement To Units of Measurement To Be Used In Air and Ground Operations, August 1990.

[26] ATC Standards Publications. *Standards for Air Traffic Controllers - Part A: Aerodrome Control.* CAP 624. Civil Aviation Authority, Civil Aviation Authority, 1E, Aviation House, Gatwich Airport South, West Sussex RH6 0YR, January 1994. ISBN 0 86039 570 7, GBP 10.

[27] ATC Standards Publications. *Standards for Air Traffic Controllers - Part B: Approach/Approach Radar Control and Area Radar Control (Aerodrome).* CAP 624. Civil Aviation Authority, Civil Aviation Authority, 1E, Aviation House, Gatwich Airport South, West Sussex RH6 0YR, February 1994. ISBN 0 86039 581 2, GBP 10.

[28] ATC Standards Publications. *Standards for Air Traffic Controllers - Part C: Area/Area Radar Control.* CAP 624. Civil Aviation Authority, Civil Aviation Authority, 1E, Aviation House, Gatwich Airport South, West Sussex RH6 0YR, January 1994. ISBN 0 86039 576 6, GBP 10.

[29] Jeff Worsinger TAB/AERO Staff. *AIM/FAR 1995 - Airman's Information Manual/Federal Aviation Regulations.* TAB AERO - Divvision of McGraw-Hill, Blue Ridge Summit, PA 17294-0850, aim revised to 18 aug 1994, far revised to 2 sept 1994 edition, 1995. ISBN 0-07-063084-4 . US $ 12.95.

[30] John A. Wise, V. David Hopkin, and Marvin L. Smith, editors. *Automation and Systems Issues in Air Traffic Control*, Berlin, Aquafredda, Italy confrence, June 1990 1991. Springer-Verlag.