Urban Planning
Processes, Documents, Management
Shanghai TongJi Lectures

Dines Bjørner
Technical University of Denmark
September 10, 2017: 12:51 and September 11, 9:30–11:15, 2017

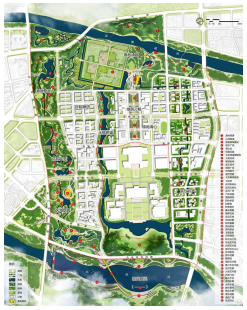
Four Lectures

First Lecture

- **Agenda:**
  - General Remarks on Urban Planning 3–9
  - Notation, an ultra-brief introduction 11–24
  - Capsule View of Urban Planning 25–32
  - Formal Methods – Why ? 33–40

1. Introductory Remarks

- Visualisations of some urban plans:




- Further visualisations of urban plans:

- And:

1.1. Urban Planning
1.1.1. "Definition"

- Urban planning is a technical and political process
  - concerned with the development and use of land,
  - planning permission,
  - protection and use of the environment,
  - public welfare, and
  - the design of the urban environment, including
    - air,
    - water, and
    - the infrastructure passing into and out of urban areas, such as transportation, communications, and distribution networks.

1.1.2. Urban Planning Information and Targets
1.1.2.1 "Input"
1.1.2.1.1. Geographic

- Geodetic
- Meteorological
- &c.
- Geotechnical
- Pre-existing plans

1.1.2.1.2. Social and Economic

- Current Welfare
- Current Economies
- &c.

1.1.2.1.3. Environmental

- Current Emissions
- &c.

1.1.2.2 "Output" Plans
1.1.2.2.1. Issues

- Cadestral
- Cartographic
- Zoning

1.1.2.2.2. Primary Plans

- Master Plan
- Shopping, Apts.
- Sports
- Industries
- Apartments
- Parks
- Office, Shopping
- Villas
- &c.

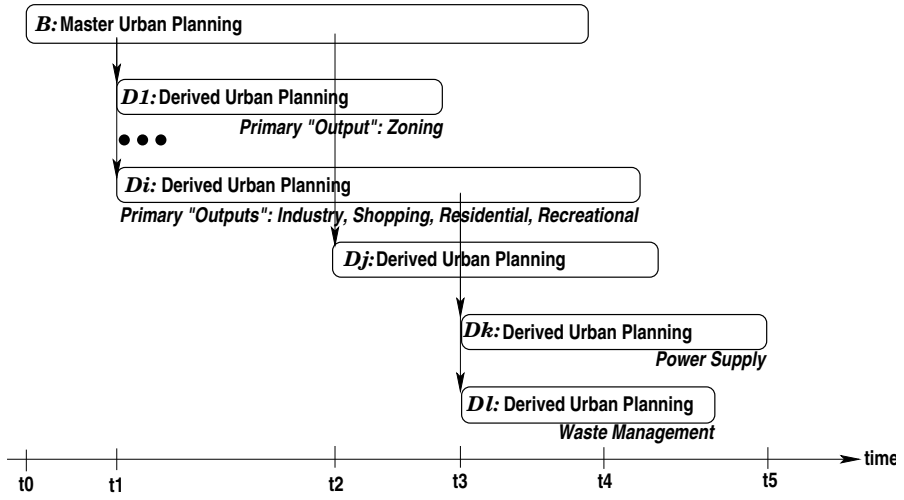
1.1.2.2.3. Secondary Plans

- Fire Brigades
- Schools
- ERs, Hospitals
- &c.

1.1.2.2.4. Infrastructure, Social &c. Plans

- Roads
- Water
- Gas
- Rails
- Sewage
- &c.
- Canals
- Electricity

1.1.3. Snapshot of Urban Planning Developments



1.2. Notation

- We usually express urban plans in drawings.
- Drawings are then expressed in some notation, maybe many!

- Talking abut 'Urban Planning'"
  must also be expressed in some notation, f.ex.:
  - natural language speech and text,
    as is basically done today[1], or
  - semi-formal texts, f.ex.: pseudo-program texts, or
  - formal texts introduced by "natural language" annotations such as we shall do!
- Therefore an ultra-brief introduction to a formal notation (i.e., "math").

[1] often "adorned" with "pie" and "hierarchical" diagrams

1.2.1. Types

- When we write:
  type
  A, B, ..., Z
  we mean that A, B, ..., Z are sets of further undefined values.
- When we write
  type
  P = Q × R
  S = W-**set**
  F = A → B (A $\xrightarrow{\sim}$ B)
  we mean that P defines the set of pairs of Q and R values,
  that S defines the set of sets of W values
  and that that F defines the set of all total (partial) functions from (some) values of type A into values of type B.
- Base types are:
  type Nat, Intg, **Bool**, **Char**, **Text**
  are predefines types.

1.2.1.1 Examples

type UAF = **Nat** → **Nat**

- UAF defines the (infinite) set of all unary arithmetic functions from natural numbers to natural numbers.
  Examples: the squaring function, the cubing function, ...

• • •

typ BAF = NAT × NAT → **Real**

- BAF defines the (infinite) set of all binary arithmetic functions from pairs of natural numbers to real numbesrs.
  Examples: addition, subtraction, multiplication, division, ...

1.2.2. Functions

1.2.2.1 Function Signatures

When we write:

value
  f: X → Y

we mean f to denote the **value** of a **total function** in the **function space** $X \to Y$,

that is, from elements in the **definition set** X

to elements in the **result set**, i.e., the **range**, Y.

We refer to f: X → Y as the **signature** (of f).

1.2.2.2 Function Invocation, i.e. Application

When we write the expression:

f(x) = y

we mean that f when **applied** to $x : X$
yields a value equal to $y : Y$.

1.2.3. Function Definitions

When we write:

value
  f(x) ≡ $\mathcal{E}$(x)

or

value
  f(x) ≡ **as** y
  pre $\mathcal{P}$(x)
  post $\mathcal{Q}$(x,y)

we mean that the function (f) value for argument x
is the value of the expression $\mathcal{E}$(x),
respectively the value y such that

the predicate expression $\mathcal{P}$(x) holds
and the predicate expression $\mathcal{Q}$(x,y) also holds.

1.2.3.1 Examples

value

square: Intg → Intg
square(i) ≡ i∗i

cube: Intg → Intg
cube(i) ≡ i∗i∗i

addition: Intg × Intg → Intg
addition(i,j) ≡ i+j

division: Intg × Intg $\xrightarrow{\sim}$ **Real**
division(i,j) ≡ i/j **pre** j≠0

1.2.4. Remarks

Function invocation takes no time!

1.2.5. Behaviours

1.2.5.1 Behaviour Signatures, I

When we write:

b: **Unit** → **Unit**

we mean that b is a **behaviour**,

which takes no arguments,
i.e., a behaviour invocation is expressed as $b()$;
goes on "forever";
and thus, yields no result!

When we write:

b: X → **Unit**

we mean that b is a **behaviour**,

which takes arguments, x in X,
i.e., a behaviour invocation is expressed as $b(x)$;
goes on "forever";
and thus, yields no result!

1.2.5.2 Behaviour Synchronisation & Communication

Behaviours synchronise & communicate via **channel**s.

One behaviour offers, on **channel** ch, the value of an expression e:

ch ! e

Another behaviour offers to accept a value on **channel** ch:

ch ?

The value of ch ? may be bound to a variable v:

let v = ch ? **in** ... $\mathcal{E}$(v) ... **end**

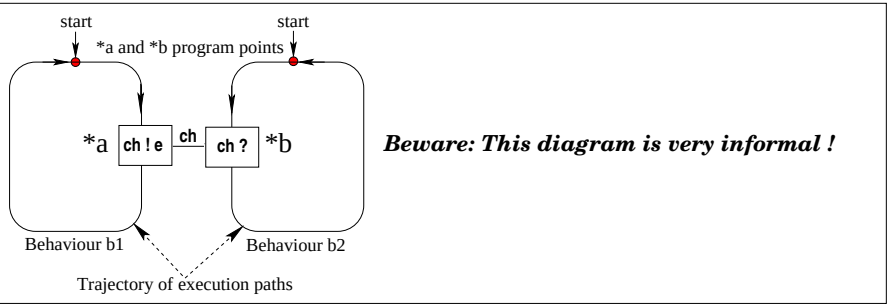
1.2.5.3 Behaviour Definitions, I

Two behaviours b_1 and b_2 may thus be defined as follows:

value
  b1(...) ≡ (...; ch ! e ; ... ; b1(...))
  b2(...) ≡ (...; **let** v = ch ? **in** ... **end**; ... ; b2(...))

Assume that the two behaviours b_1 and b_2 occur simultaneously:

  b1 ‖ b2



Beware: This diagram is very informal !

We assume an operational understanding of the two processes.

There are the following possibilities:

- Either b_1 "arrives" at program point $*a$ before b_2 arrives at $*b$.
  - Then b_1 waits.
  - When b_2 "arrives" at program point $*b$ both behaviours synchronise and b_1 communicates value of e to b_2.
- Or b_2 "arrives" at program point $*b$ before b_1 arrives at $*a$.
  - Then b_2 waits.
  - When b_1 "arrives" at program point $*a$ both behaviours synchronise and b_2 accepts value of e.
- Etcetera.

1.2.5.4 Channels

Channels are declared:

channel ch M

ch can now be referred to in any behaviour signature and definition.

If in a clause ch ! e then e must be of type M.

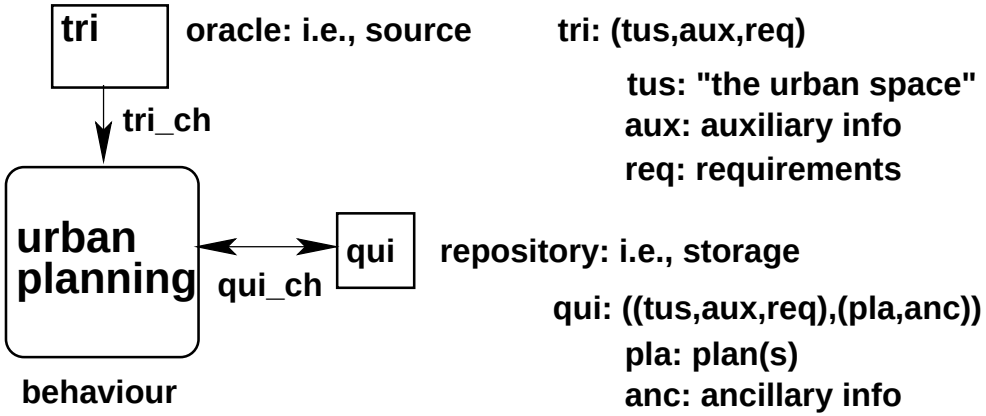
In in an expression ch ? then that expression is of type M.

1.2.5.5 Behaviour Signatures, II

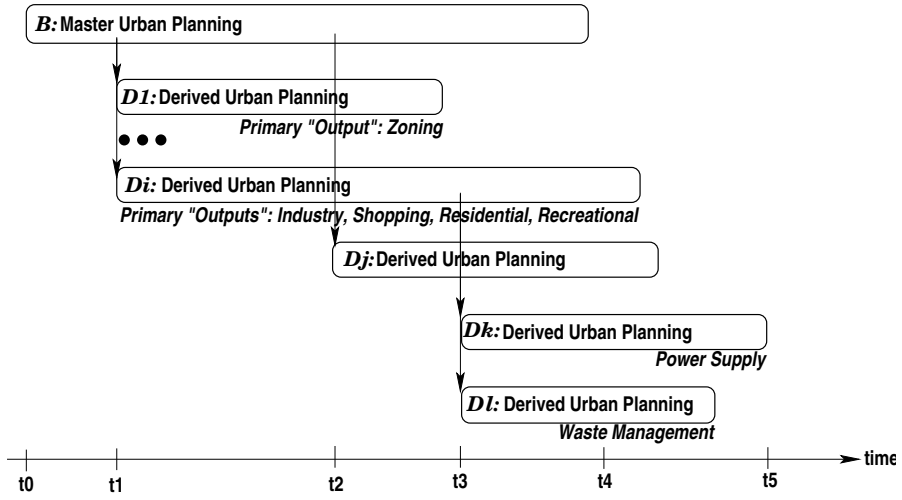
The signatures of the two behaviours b_1 and b_2 are

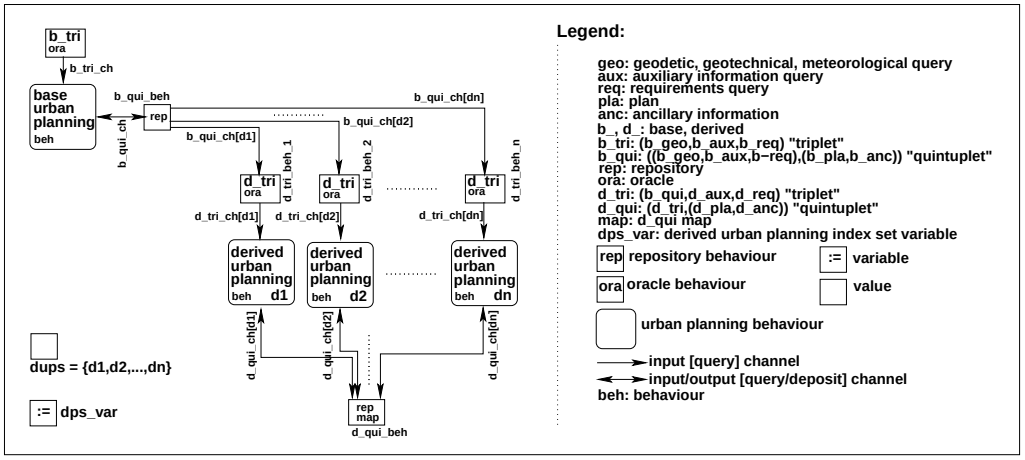
value
  b1: A → **out** ch **Unit**
  b2: B → **in** ch **Unit**

1.3. Capsule View of Urban Planning
1.3.1. "The Base Urban Planning"



1.3.2. "The Master and Derived Urban Plannings"





1.3.3. "The Data"

- Very simplistically:
  - "Input"

type	
1	GEO
2	AUX
3	REQ
6	TRI = GEO×AUX×REQ

  - "Output"

type	
4	PLA
5	ANC
7	RES = PLA×ANC
8	QUI = TRI×RES

1.3.4. Function and Behaviour
1.3.4.1 Function, I

```
type
6  TRI = GEO×AUX×REQ
8  QUI = TRI×RES
value
14  up_fct: TRI → QUI → QUI
15  up_fct(tri)(qui) as (tri′,(pla,anc))
16      tri = tri′ ∧
17      P_base(tri′)(qui)(pla,anc)
```


1.3.4.2 The TRI Oracle

```
type
6  TRI = GEO×AUX×REQ
channel
18  tri_ch:TRI
value
19  tri_beh: TRI → out tri_ch Unit
19  tri_beh(tri) ≡
20      let tri′:TRI · tri_fit(tri,tri′) in
21      tri_ch ! tri′ ;
22      tri_beh(tri′)
19      end
24  pre: tri_fit(tri,tri′)
```


1.3.4.3 The QUI Repository

```
type
8   QUI = TRI×RES
channel
26   qui_ch:QUI
value
27   qui_beh: QUI → in,out qui_ch Unit
27   qui_beh(qui) ≡
29       qui_beh(qui_ch?)
28       ⫿
30       qui_ch!(qui) ; qui_beh(qui)
```


1.3.4.4 Behaviour, I

```
value
31  base_up_beh_0: Unit → in tri_ch out qui_ch Unit
32  base_up_beh_0() ≡
33      let (tri,qui) = (tri_ch?,qui_ch?) in
34      let qui = base_up_fct(tri)(qui) in
35      qui_ch ! qui end end ;
36      base_up_beh_0()
```

1.4. Formal Methods – Why ?
1.4.1. Method and Methodology
1.4.1.1 Method

- By a **method** we shall understand
  - a set of **principles** for **selecting** and **applying**
  - a number of **techniques**
  - and **tools**

  in order to construct an artifact.

1.4.1.2 Methodology

- By **methodology** we shall understand
  - the **study** and **knowledge** about **method**s.

1.4.2. Formal Methods

- By a **formal method** we shall understand
  - a **method** [some of] whose
  - **techniques** and **tools**
  - has a **mathematical foundation**.

1.4.3. Formal Specification Language

- By a **formal specification language** we shall understand
  - a **language** whose
    - **syntax** and
    - **semantics**
  - has a **mathematical foundation**
  - and which has a **formal proof system**

  which enables us to prove properties of specificatioms.

1.4.4. Principles
1.4.4.1 The Triptych Principle

- A major principle of ours for developing software is embodied in the **Triptych** method:
  - Before **software** can be developed we must understand its **requirements**.
  - Before requirements can be expressed we must understand the **domain**.

- So we proceed in three, more-or-less consecutive phases:
  - **domain engineering**,
  - **requirements engineering** and
  - **software design**.
- verifying (validating, proving, testing) properties of
  - the **domain**,
  - the **requirements**,
  - the **domain-to-requirements** "translation",
  - the **software** and the
  - the **requirements-to-software** "translation".

1.4.5. Our Work on 'Urban Planning' is, so far, Domain Engineering

- We must describe what *'urban planning'* is.
- We must understand "all" facets of urban planning
- We must be able precisely to communicate what urban planning is.
- We must be able to separate concerns of urban planning into
  - those that have to do with project management,
  - those that have to do with data management,
  - those that really have to do with urban planning, and
  - "all the other things" !

. But is Necessary for Requirements Engineering and Software Desigr

- Process Management
- Data Management

1.4.6.1 The Domain Engineering Principle [Hoare]

1 The models describe all aspects of the real world that are relevant for any good software design in the area. They describe possible places to define the system boundary for any particular project.

2 They make explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.

3 They describe the whole range of possible designs for the software, and the whole range of technologies available for its realisation.

4 They provide a framework for a full analysis of requirements, which is wholly independent of the technology of implementation.

5 They enumerate and analyse the decisions that must be taken earlier or later in any design project, and identify those that are independent and those that conflict. Late discovery of feature interactions can be avoided.

1.4.7. Why Mathematics, i.e., Formal Methods ?

Mathematics is there –
to be used, as in all branches of the Natural Sciences.

- It makes it easy to precisely pin-point and "limit" areas of concern.
- Formalising a(ny) domain makes it possible to express precisely whether the model is complete and consistent.
- It enables systematic testing, model checking and formal verification
of possibly "derived" software.
- A formal, complete and consistent domain model
expresses that we have truly understood the domain.

Informal-understanding-only of a domain does not allow the •s.

2. The Next Days

- Tomorrow and Wednesday I will cover the report:
  - **Urban Planning Processes**: **A Research Note**

  systematically.
  - Tomorrow I will cover aspects of *Master Planning*
  - Wednesday I will cover *Derived Urban Planning*s

Thank You

3. Base Urban Planning

Second Lecture

- **Agenda:**

3.1. Urban Planning Information Categories

3.1.1. "Input"

- Among the arguments of urban planning are

1 information, bTSU, about *the urban space,* the demo-geographic area subject to planning: its geodetic "make-up", its geotechnical and meteorological properties, etc.[2]

2 related, but not geographic, information, bAUX[iliary][3],

3 and some requirements, bREQ[uirements].

type
1 bTSU
2 bAUX
3 bREQ

[2] We shall not include information about the use of the existing land. We refer to Slide 49 for how we handle such information.
[3] Auxiliary: giving help or support.

3.1.2. "Output"

Among results of urban planning are

4 "the plan" (or "plans"), bPLA[ns],

5 and possibly some other ancillary[4] documents, bANC[illerary].

type
4 bPLA
5 bANC

[4] Ancillary: providing necessary support to the main work

3.2. The Iterative Nature of Urban Planning

- We take it that urban planning proceeds in "cycles":

6 In each cycle the base urban planning function, base_up_fct, is applied to an input argument triple, (b_tus,b_aux,b_req):(bTUS×bAUX×bREQ):bTRI, of "fresh" geodetic/geotechnical/meteorological (etc.), auxiliary and requirements information.

type
6 bTRI = bTUS × bAUX × bREQ

7 Each cycle, that is, each application of base_up_fct, results in a "most recent", not necessarily "final", plan and ancillary information, (b_pla,b_anc):bPLA×bANC:bRES.

type
7 bRES = bPLA × bANC

8 But, to "drive" the urban planning process, base_up_beh, towards "final", that is, an adequately satisfactory plan etc., the urban planning function, base_up_fct,

- *need also be provided with the results of the previous iteration's result* —
- which we take to be a ("quintuplet") pair
- of an (i.e., the "previous") "input" triple
- and the previous result pair.

type
8 bQUI = bTRI × bRES

3.3. Initialisation

Urban planning proceeds in iterating from initial

9 urban space, auxiliary and requirements information, as well as

10 (usually "empty") plans and ancillaries.

We extend the notion of initial values to

11 triplet arguments,

12 result pairs, and

13 "quintuplet" argument/result pairs.

towards such results (plans and ancillaries) that are deemed satisfactory.

value
9 b_tus_init: bTUS,
9 b_aux_init:bAUX,
9 b_req_init:bREQ
10 b_pla_init: bPLA,
10 b_anc_init:bANC
11 b_tri_init: bTRI = (b_tus_init,b_aux_init,b_req_init)
11 **assert:** b_tri_fit(b_tri_init,b_tri_init)
12 b_res_init: bRES = (b_pla_init,b_anc_init)
13 b_qui_init: bQUI = (b_tri_init,b_res_init)

- We refer to Item 23 on Slide 56 for an explanation of the b_tri_fit predicate.

3.3.1. Existing versus Evolving Plans

- The quintuplet "feedback",
  - which includes a 'plan' component,
  - secures that possibly pre-existing plans
  - are included, as initialised components of the plan results.[5]
- The iterative nature of urban planning
  - thus allows for stepwise urban re-development,
  - from existing "urban land-scapes"
  - via mixed "previous" and "future" land-scapes
  - to the final urban development plan.

[5]We refer to Item 1 on Slide 42 and footnote 2 on Slide 42.

3.4. A Simple Functional Form

14 The base urban planning *function*, base_up_fct, thus applies to
- (i) a "most recent" triplet of urban space, auxiliary and requirements information, and to
- (ii) a "past quintuplet", a resumption[6], that is, pair of urban space, auxiliary and requirements information as well as plan and ancillary information and yields such a resumption "quintuplet" pair of a triplet and a pair.

15 The application of base_up_fct to such arguments, i.e.,
- base_up_fct(b_tus,b_aux,b_req)(b_qui)
- yields a "quintuplet" result, a resumption,
- b_qui:((b_tus′,b_aux′,b_req′),(b_pla,b_anc)).

[6]Resumption: like a repetition, a continuation

We "explain" the relations between "input" arguments and "output" (**as**) results:

16 The "input" argument b_tri
is "carried forward", b_tri′ (=b_tri),
to be redeposited as part of the result.

17 The main part of the result, (b_pla,b_anc),
is related, $\mathcal{P}$, to the input argument
including the previous "result", the resumption.

```
14  base_up_fct: bTRI → bQUI → bQUI
15  base_up_fct(b_tri)(b_qui) as (b_tri′,(b_pla,b_anc))
16      b_tri = b_tri′ ∧
17      P_base(b_tri)(b_qui)(b_pla,b_anc)
```

- For the time being we shall leave the base urban planning function, base_up_fct, that is, $\mathcal{P}_{\text{base}}$, uninterpreted.

3.5. Oracles and Repositories

- Oracles are simple behaviours
  - that *offers* information to other behaviours.
- Repositories are simple behaviours
  - that *store* information from other behaviours
  - and *offers* stored information to other behaviours.

3.5.1. The Base 'Input' Oracle

- An urban planning oracle, when so requested, will select some information – usually in some non-deterministic fashion, and usually subject to some constraint – and present this information to the requestor, i.e., an urban planning behaviour.
- In this section we shall deal with one specific oracle, b_tri_beh:
  - one that "assembles" triplets, b_tri,
    - of urban space, b_tus:bTUS,
    - auxiliary, b_aux:bAUX, and
    - requirements, b_req:bREQ,

    information
  - to requesting behaviours.

We introduce a pair of specification components:

18 a *channel*, b_tri_ch,
- over which a base urban planning behaviour, base_up_beh,
- offers to receive triplets, b_tri:bTRI,
- from an oracle, b_tri_beh,

19 and an *oracle*, b_tri_beh,
which "remembers" its most recently communicated triplet[7].

channel
18 b_tri_ch:bTRI
value
19 b_tri_beh: bTRI → **out** b_tri_ch **Unit**

[7] The oracle is initialised with b_tri_beh(geo_init,b_aux_init,b_req_init).

20 The oracle assembles (b_tri′:bTRI),
a base triplet which satisfies a predicate b_tri_fit(b_tri,b_tri′).

21 That triplet is offered,
b_tri_ch ! b_tri′,
to the base urban behaviour –

22 whereupon the oracle resumes being the oracle, now, however, with the recently assembled base triplet as its resumption.

```
19  b_tri_beh(b_tri) ≡
20      let b_tri′:bTRI · b_tri_fit(b_tri,b_tri′) in
21      b_tri_ch ! b_tri′ ;
22      b_tri_beh(b_tri′)
19      end
24  pre: b_tri_fit(b_tri,b_tri)
```


23 The fitness predicate, b_tri_fit(b_tri,b_tri′),
checks whether a "newly" assembled base triplet, b_tri′,
stands in some suitable[8] relation $\mathcal{P}$(b_tri,b_tri′) to a a similar (f.ex., "earlier") base triplet, b_tri.

24 The fitness predicate holds for b_tri_fit(b_tri,b_tri).

```
23  b_tri_fit: bTRI × bTRI → Bool
23  b_tri_fit(tri,tri′) ≡ P(b_tri,b_tri′)
```

25 The oracle, b_tri_beh, is initialised with an initial triplet value b_tri_init, cf. formula Item 11 on Slide 47.

```
23  b_tri_beh(b_tri_init): assert: b_tri_fit(b_tri_init,b_tri_init)
```

[8] – to be defined for each specific urban planning project

3.5.2. The Base Resumption Repository

- The "quintuplet" pair of
  - an "input" triple
  - and a result pair,
  - b_qui:(b_tri:bTRI,(b_pla:bPLA,b_anc:bANC))
- is thought of as residing in a *repository* behaviour, b_qui_beh,
  - which "receives" (b_qui_ch?) "quintuplets" from the urban planning behaviour,
  - or "offers" (b_qui_ch!b_qui) such to the urban planning behaviour.

26 There is therefore a channel, b_qui_ch, between the urban planning behaviour and the "quintuplet" repository behaviour,

27 b_qui_beh.

28 It either

29 accepts or

30 offers quintuplets.

```
channel
26   b_qui_ch:bQUI
value
27   b_qui_beh: bQUI → in,out b_qui_ch  Unit
27   b_qui_beh(b_qui) ≡
29       b_qui_beh(b_qui_ch?)
28       []
30       b_qui_ch!(b_qui) ; b_qui_beh(b_qui)
```


3.6. A Simple Behavioural Form

- Urban planning, however, is a time-consuming "affair".
- So we model it as a behaviour.

31 The base_up_beh_0[9] behaviour

- takes no argument, hence the left signature element: **Unit**,
- avails itself of the input channel for obtaining
  - proper input, b_tri,
  - and b_qui,

  for the base urban function, base_up_fct,
- and output channel, for depositing a resumption, b_qui′,
- and (then) "goes on forever", as indicated by the right signature element: **Unit**.

```
31   base_up_beh_0: Unit → in b_tri_ch out b_qui_ch Unit
```

[9] As there will be several versions, from simple towards more elaborate, of the base_up_beh behaviour, we index them.

32 The simple (version of the) base_up_beh_0 behaviour

33 obtains the base triplet, b_tri, and the base resumption, b_qui,

34 performs the base_up_fct planning function and

35 provides its result, a resumption, b_qui′, to the quintuplet repository,

36 whereupon it reverts to being base_up_beh_0.

```
value
32   base_up_beh_0() ≡
33       let (b_tri,b_qui) = (b_tri_ch?,b_qui_ch?) in
34       let b_qui′ = base_up_fct(b_tri)(b_qui) in
35       b_qui_ch ! b_qui′ end end ;
36       base_up_beh_0()
```

- The base_up_beh_0 behaviour
  - repeatedly "performs" urban planning, "from scratch",
  - as if new urban space, auxiliary and requirements information was "new" in every re-planning
  - — "ad infinitum" !
- We now revise base_up_beh_0 into base_up_beh_1
  - — a behaviour "almost" like base_up_beh_0,
  - but one which may terminate.

37 base_up_beh_1

38 first behaves like base_up_beh_0 (Items 33–35)

39 then checks whether the obtained base resumption is satisfactory, that is, is OK as an end-result of base urban planning.

40 If so then base_up_beh_1 terminates,

41 else it resumes being base_up_beh_1.

```
value
31  base_up_beh_1: Unit → in b_tri_ch out b_qui_ch Unit
37  base_up_beh_1() ≡
38      let b_qui = b_base_up_fct(b_tri_ch?)(b_qui_ch?) in b_qui_ch!b_qui ;
39      if b_qui_satisfactory(b_qui)
40          then skip
41          else base_up_beh_1() end
37      end

39  b_qui_satisfactory: bQUI → Bool
```


- The b_qui_satisfactory predicate
  - inquires the base quintuplet, b_qui, as for its suitability as a final candidate for an urban plan[10].

[10] The b_qui_satisfactory argument, b_qui, embodies not only that plan, but also the basis for its determination.

3.7. That was all for today !

- Tomorrow I will cover a generalisation of urban planning.

4. Derived Urban Plannings

Third Lecture

- **Agenda:**
  - Preliminaries 66–73
  - The Derived Urban Planning Functions 74–77
  - The Derived Urban Planning Behaviours 78–78
  - The Derived Resumption Repository 79–82
  - A Visual Rendition of Urban Planning 83–87
  - The Urban Planning System 88–93

4.1. Preliminaries

4.1.1. Derived Urban Plan Indices

- We think of *base* urban planning function, modeled by base_up_fct,
  - as being concerned with the overall "division" of the urban space (i.e., geographical area, land and water) into zones for building, recreation, and other (i.e., the ***master plan***).
  - Aggregations of these zones, one, more or all (usually several), can then be further "[derive] planned" into
    - (d_1) light, medium and heavy industry zones,
    - (d_2) mixed shopping and residential zones,
    - (d_3) apartment bldg. zones,
    - ..., etc., etc.,
    - (d_{m-1}) villa zones, and
    - (d_m) recreational zones.

  - Additional forms of derived plannings are:
    - (d_{m+1}) transport,
    - (d_{m+2}) electricity supply,
    - (d_{m+3}) water supply,
    - (d_{m+4}) waste management,
    - (d_{m+5}) health care,
    - (d_{m+6}) fire brigades,
    - ..., etc., etc.,
    - (d_{m+n}) schools.
- We refer to the d_i's as derived urban plan indices.

42 We think of this variety of "derived" plannings as indexed such as hinted at above,

43 and dups as the set of all indices.

```
type
42  DP == {|d1,d2,...,dn|}
value
43  dups:DP-set = {d1,d2,...,dn}
```

4.1.2. A "Reservoir" of Derived Urban Planning Indices

44 To secure that at most one derived planning, d_i (per i), is initiated

- we introduce a global variable, dps_var,
- initialised to an empty set of derived planning tokens
- and updated with the addition of selected DP tokens.

```
variable
44    dps_var:DP-set := {} comment dps_var denotes a reference
```


4.1.3. A Derived Urban Planning Index Selector

45 A function, sel_dps, selects zero, one or more "fresh" DP indices, that is, DP tokens that have not been selected before.

```
value
45    sel_dps: Unit → DP-set
45    sel_dps() ≡
45        let dps:DP-set·dps⊆dups \ c dps_var in
45        dps_var := c dps_var ∪ dps; dps end
comment
44    [ c denotes a contents-taking operator ]
```

- We shall revise the above selector in on Slide 90.

4.1.4. Let us recall:

```
value
31    base_up_beh_1: Unit → in b_tri_ch out b_qui_ch Unit
37    base_up_beh_1() ≡
38        let b_qui = b_base_up_fct(b_tri_ch?)(b_qui_ch?) in b_qui_ch!b_qui ;
39        if b_qui_satisfactory(b_qui)
40            then skip
41            else base_up_beh_1() end
37        end
```


4.1.5. The Derived Urban Plan Generator

46 We therefore edit the base_up_beh_1 behaviour slightly into the revised base_up_beh_2.

- In base_up_beh_2 we insert, "in parallel" (‖) with the "resumption" of base_up_beh_2,
- an internal non-deterministic choice behaviour, der_up().
- It specifies the selection of zero, one of more DP tokens,
- and initiates corresponding derived planning behaviours, $\mathsf{der\_up\_beh}_i()$,
- as well as their corresponding "input" triplet oracles, $\mathsf{d\_tri\_beh}_i()$.
- But only at most once.

47 These derived planning behaviours, $\mathsf{der\_up\_beh}_i$, and "input" triplet oracles, $\mathsf{d\_tri\_beh}_i()$

- are like base_up_beh_1, respectively b_tri_beh,
- only now they are "tuned" to the specific derived planning issues (i.e., $_i$).

```
value
46    der_up: Unit → Unit
46    der_up() ≡ let dps = sel_dps() in ‖{der_up_beh_i()‖d_tri_beh_i()|i:DP·i ∈ dps} end
```

- We shall introduce the $\mathsf{der\_up\_beh}_i$ and $\mathsf{d\_tri\_beh}_i$ behaviours below.

4.1.6. The Revised Base Urban Planning Behaviour

- We "take over" the basic structure and definition ("contents") of the urban planning function and behaviour from that of the base versions.

48 We think of zero, one or more derived plannings (der_up_beh$_1$, der_up_beh$_2$, ..., der_up_beh$_n$) being initiated after some stage of *base* function, base_up_fct, has concluded.

```
value
37''  base_up_beh_2() ≡
41''     let b_qui=base_up_fct(b_tri_ch?)(b_qui_ch?) in b_qui_ch!b_qui ;
39''     if base_satisfactory(b_qui)
37''        then skip
38''        else
48             der_up() ||
40''           base_up_beh_2() end end
```


4.2. The Derived Urban Planning Functions

- An important form of information for each derived urban planning function
  - is the resumption, i.e., the quintuplet information
  - from the base urban behaviour:
  - bQui.

49 The new forms of information are:

- the derived urban planning auxiliary, dAUX$_i$,
- the derived urban planning requirements information, dREQ$_i$,
- as well as the derived urban planning plans, dPLA$_i$,
- and their ancillary information, dANC$_i$.

50 The primary arguments for the derived urban planning function, base_up_fct, is therefore a "quintuplet", d_qui:dQUI, of

- a base triplet, b_tri:bTRI, and
- the pair of
  - the derived urban planning auxiliary information, d_aux$_i$:dAUX$_i$,
  - and the derived urban planning requirements, d_req$_i$:dREQ$_i$.

The result of derived urban planning function, der_up_fct, as for the base urban planning function, base_up_fct,

51 is that of a "quintuplet", also a resumption, dQUI$_i$, of the primary arguments, b_tri:bTRI, and

52 the result, a pair of a derived plan, d_pla$_i$, and derived ancillaries, d_anc$_i$.

53 As for the base urban planning function, base_up_fct, it has a secondary, derived "quintuplet" argument (which, as for base_up_fct, helps "kick-start" urban planning). This second argument is the result of a previous application of the der_up_fct.

54 The derived urban planning function der_up_fct$_i$ signature is therefore that of a function from a triplet of a most recent base quintuplet, derived urban planning auxiliary and derived urban planning requirements information to functions from derived "quintuplet" arguments to derived "quintuplet" results.

55 The triplet argument, d_tri$_i$, and the first part of the result, also a triplet, d_tri$'_i$, are the same.

56 The derived urban planning function der_up_fct$_i$ is further characterised by a predicate, $\mathcal{P}_{\mathsf{der}_i}$, which we leave further undefined.

type

49 dAUX$_1$, dAUX$_2$, ..., dAUX$_n$
49 dREQ$_1$, dREQ$_2$, ..., dREQ$_n$
49 dPLA$_1$, dPLA$_2$, ..., dPLA$_n$
49 dANC$_1$, dANC$_2$, ..., dANC$_n$
50 dTRI$_i$ = bQUI×dAUX$_i$×dREQ$_i$ [i:DP·i∈dups]
52 dRES$_i$ = dPLA$_i$ × dANC$_i$ [i:DP·i∈dups]
51 dQUI$_i$ = dTRI$_i$×dRES$_i$ [i:DP·i∈dups]

value

53 der_up_fct$_i$: dTRI$_i$ → dQUI$_i$m → dQUI$_i$ i:DP
54 der_up_fct$_i$(d_tri$_i$)(d_qui$_i$) **as** (d_tri$'_i$,d_res$_i$)
55 d_tri$_i$ = d_tri$'_i$ ∧
56 $\mathcal{P}_{\text{der}_i}$(d_tri$'_i$,d_res$_i$)

56 $\mathcal{P}_{\text{der}_i}$: dTRI$_i$ × dRES$_i$ → **Bool**

4.3. The Derived Urban Planning Behaviour

57 We think of zero, one or more derived plannings (der_up_beh$_{i_1}$, der_up_beh$_{i_2}$, ..., der_up_beh$_{i_m}$) being initiated after some stage of the *der_up_fct$_i$* function has concluded.

value

37''' der_up_beh$_i$() ≡
41''' **let** d_qui$_i$ = der_up_fct$_i$(d_tri_ch[i]?)(d_qui_ch[i]?) **in**
41''' d_qui_ch[i] ! d_qui$_i$;
39''' **if** der_satisfactory$_i$(d_qui$_i$)
37''' **then skip**
38''' **else**
57 der_up() ||
40''' der_up_beh$_i$() **end end**

4.4. The Derived Resumption Repository

4.4.1. The Consolidated Derived Resumption Map

58 The derived urban planning functions (and thus behaviours) operate, not on simple resumptions, as do the base urban planning functions (and behaviours), but on the aggregation of all derived functions' (etc.) quintuplets, that is, an indexed set of quintuplets – modeled as a derived resumptions map.

type

58 dQUIm = DP $\xrightarrow[m]{}$ dQUI$_i$

4.4.2. The Consolidated Derived Resumption Repository Channel

59 Communications between the individual derived urban planning behaviours and the consolidated derived resumption repository are via an indexed set of channels communicating derived resumptions maps.

channel

59 {d_qui_ch[i]:dQUIm|i:DP· i ∈ dups}

4.4.3. The Consolidated Derived Resumption Repository

60 The consolidated derived resumption repository behaviour either ([]) updates its state map with received individual derived resumptions, or offers the entire such state maps to whichever derived urban planning behaviour so requests.

```
value
60  d_qui_beh: dQUIm → in,out der_qui_ch[i] Unit  i:DP
60  d_qui_beh(d_qui_m) ≡
60    (d_qui_beh(d_qui_m†[i↦d_qui_ch[i]?])
60     []
60     d_qui_ch[i]!(d_qui_m)) ;
60    d_qui_beh(d_qui_m)
```


4.4.4. Initial Consolidated Derived Urban Plannings

```
value
  d_qui_m = [ d1 ↦ init_d_qui1, ..., dn ↦ init_d_quin ]
```

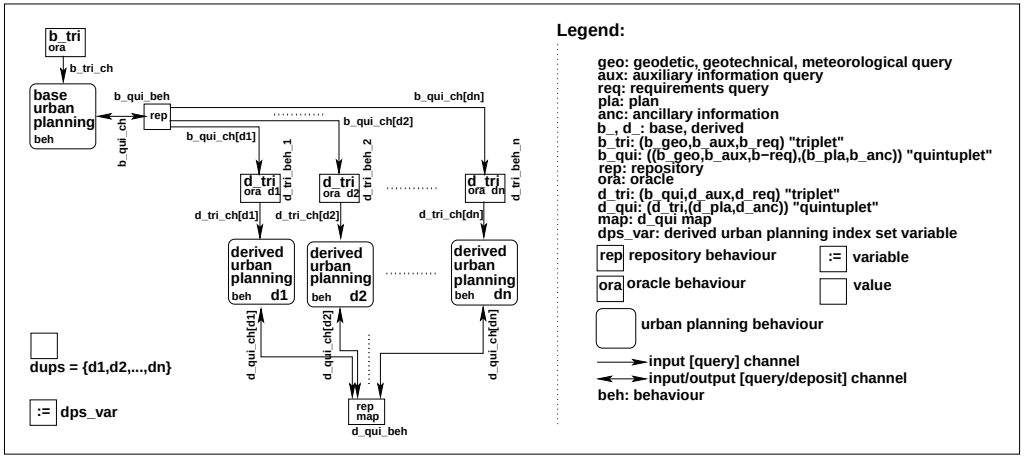
4.4.5. Initialisation of The Derived Quintuplet Oracle

- As for base oracle and repository behaviours we initialise the derived quintuplet oracle:

```
der_qui_beh(init_d_qui_m)
```


4.5. A Visual Rendition of Urban Planning Development

- The urban planning domain, when "operating at full speed", consists of
  - the base urban planning behaviour (i.e., project),
  - zero, one or more derived urban planning behaviours, each of the latter initiated by either
    - the base urban planning project or
    - a derived urban planning project.



- The planning behaviours, both the base and the derived, invoke respective urban planning functions, and these produce, such as we have modeled them, **qui**ntuplets of information, which are deposited with respective **qui**ntuplet repository behaviours:
  - the base **qui**ntuplet repository behaviour, and
  - the derived **qui**ntuplet repository behaviour — which maintains these quintuplets for all (invoked and thus ongoing) derived urban planning projects.

- We kindly ask you to review the figure given on Slide 84.
  - All you have to 'master' is the fact that there is
    - one base urban planning project, with its repository of base urban planning "quintuplets", and
    - between 0 and n derived urban planning projects, with their shared, derived urban planning "quintuplets",

    - Then there are the channels: the query (input) channels
      - providing auxiliary and requirements information to both the one base urban planning project and
      - the n derived urban planning projects;
    - and the query/repository channels
      - providing "quintuplet" aggregated information to the base urban planning project,
      - as well as "quintuplet" aggregated information to the derived urban planning projects.
  - Finally there are the "global"
    - value representing the index set of derived urban planning indices, and
    - variable which holds the index set of derived urban planning indices of ongoing derived urban planning projects.

4.6. Revised Selection of Derived Urban Plannings

4.6.1. Review

- The derived urban planning generator function, der_up, cf. Item 46 on Slide 72,

```
value
46  der_up: Unit → Unit
46  der_up() ≡
46      let dps = sel_dps() in
46      ‖ {der_up_beh_i()‖d_tri_beh_i()|i:DP·i ∈ dps} end
```

- was invoked with no arguments, der_up(),
- cf. Item 48 on Slide 73 and Item 57 on Slide 78

```
48          der_up() ‖ [ respectively ]
57          der_up() ‖
```

4.6.2. A Potential Derived Urban Plan Indices Selector

- Selection of potential derived urban planning indices was therefore rather arbitrary.
- We now let the selection depend on the aggregated resumption state of all (ongoing and) derived urban planning behaviours.

61 Function sel_dups examines either the base resumption or the aggregated resumption state of all (ongoing and) derived urban planning behaviours and yields a set of derived urban planning indices.

62 How it does that is, of course, not defined here.

```
value
61  sel_dups: (bQUI|dQUIm) → DP-set
62  sel_dups(dquim) ≡ ...
```


4.6.3. A Revised Derived Urban Plan Index Set Selector

63 We revise the derived urban plan index selector function give earlier, cf. Item 45 on Slide 70. A function, sel_dps, selects zero, one or more DP "fresh" indices, that is, DP tokens that have not been selected before.

```
value
63  sel_dps: DP-set → DP-set
63  sel_dps(dups) ≡
63      let dps:DP-set·dps⊆dups∩dups \ c dps_var in
63      dps_var := c dps_var ∪ dps; dps end
```


4.6.4. Revision of Derived Urban Plan Invocation

- We need to revise the two occurrences of der_up() –
  - in the base urban planning behaviour, and
  - in the (indexed set of) derived urban planning behaviours.
- Thus

```
48          der_up() || [ respectively ]
57          der_up() ||
```

- is to be replaced by:

```
48          der_up(b_qui_ch?) || ... [ respectively ]
57          der_up(d_qui_ch[ i ]?) || ...
```


4.7. The Urban Planning System

64 Finally we can define an urban planning development as a system of concurrent behaviours:

- the base urban planning behaviour,
- the base "quintuplet" repository and
- the derived and consolidate "quintuplet" repository

```
value
64  up_sys: Unit → Unit
64  up_sys() ≡ base_up_beh() || b_qui_beh(b_qui_init) || d_qui_beh(d_qui_m)
```

- Recall that
  - the derived urban planning behaviours as well as
  - the derived triplet behaviours

  are started by the base as well as the derived urban planning behaviours.

This was all for today !

- On Tuesday 19 September we shall discuss
  - what has been achieved,
  - what needs to be done

  and I will present
  - the beginnings of a formal model of the information type TUS.

Last Lecture

- **Agenda:**

5. Urban Planning Management

- The description we have given of urban planning emphasizes an iterative nature of urban planning.
- Individual urban planning behaviours, such as we describe it, alternate between many, more-or-less consecutive actions:
  - gathering ("reading") information for urban planning functions;
  - actually applying urban planning functions;
  - possibly starting new, derived urban planning behaviours while
  - collecting ("writing") function results.
- The base and derived urban plannings may thus involve hundreds of urban planning actions
  - their timely interaction (via oracles and repositories)
  - and consistent "production" and "use" of information.

- The urban planning description makes precise these interactions, "productions" and "usages".
  - Thus the described urban planning development process model can serve as a basis for a computerised project management support system.
- Thus, from the description we can
  - "refine" the requirements for such software,
  - and we can then code the software design.

This is a major project.

6. Data Management

- The description we have given of urban planning emphasizes also, one can say, separately,
  outlines major classes of information, i.e., data:
  - their iterated "production" and "use";
  - their "oracled" origin or
  - their "reposited" storage.
- These classes of information, and there are many,
  - need be formalised
  - and related.

- Thus the description may also serve as a basis for the
  - the development of requirements
  - and subsequent development of software

  for an *Urban Planning Database System*.

This is [also] a major project.

7. Validation & Verification

7.1. Validation of a Specification of a Product

- Validation of a specification of a product is a process and a document which
  - ensures that the document specifies the right product,
  - that is, that the customer gets what is expected,
  - and that the process leads to such a document.

7.2. Verification of a Specification of a Product

- Verification of a specification of a product is a process and a document which
  - ensures that the document specifies the product right,
  - that is, that the product is correct, ie., does not fail,
  - and that the process leads to such a specification (for us: software).

7.3. V&V of the Domain Description

- So we need to formalise first our understanding of what we mean by:
  - geodetic,
  - geotechnic,
  - meteorology,
  - social,
  - economic,
  - auxiliary,
  - plan, and
  - ancillary

  information.
- Only then does it make sense to express what we mean by
  - validating and
  - verifying

  the model.

7.4. V&V of an Urban Planning Project

- Similarly !

8. Discussion

8.1. What Have We Done ?

- Identification of Urban Planning Information Categories
- A Process Model for Urban Planning and its Formalisation
- Identification of Base- and Derived Urban Plannings
- Focus of an Iterative Nature of Urban Planning
- The Process Model as
  A Basis for Urban Planning Management
- The Information Categories as
  A Basis for Urban Planning Data Management

8.2. What Have We Not Done !

- We have not Formalised the Urban Planning Information Categories.
  - ⟐ In Sect. **??** we make a first attempt !
- We have not described the Urban Planning Functions, at all !

8.3. What Can I Do in September 2017 ?

- Make more precise the Categories of Urban Planning Information[11]
- Sketch initial Formalisations of some of these.
- An attempt is made as from Slides 106 on ...
- Validate or refute Aspects of the Urban Planning Process Model
- Make more precise some of the Urban Planning Functions
- Sketch initial Formalisations of some of these

. . . . . .

[11] *Please: I wish to examine available example of all the eight kinds of urban planning information listed in the first • on Slide 100.*

8.4. Further to Be Done

- Requirements for Urban Planning Project Management Software
- Requirements for Urban Planning Data Management Software

9. Closing Words

Thanks for inviting me to TongJi!

10. An Attempt at a Formalisation of "The Urban Space"

10.1. Main Parts

- To the left, in the framed box, we **narrate** the story.
- To the right, in the framed box, we **formalise** it.
- One way of observing *the urban space* is presented:

65 We can speak of The Urban Space, TUS, in terms of its	**type** 65 TUS, GeoD, GeoT, Met, Soc, Eco, ...
66 GeoDecy (i.e., geodetic features),	**value** 66 obs_GeoD: TUS → GeoD
67 GeoTechniques,	67 obs_GeoT: TUS → GeoT
68 Meteorology,	68 obs_Met: TUS → Met
69 Social features,	69 obs_Soc: TUS → Soc
70 Economic features, etcetera.	70 obs_Eco: TUS → Eco

- The obs_P: M → P is the **signature** of a postulated *(observer) function.*
- From parts of type M it **observes** [sub-]parts of type P.

10.2. Attributes

10.2.1. Urban Space Attributes – Informal

- Attributes are also called *properties*, *qualities* or *indicators*.
- We list some urban space **attributes**:
  - ◈ *Geodetic:*
    - ◎ land elevation (isometric lines etc.)
    - ◎ water: springs, creeks, rivers, lakes, oceans; dams, canals, ...
    - ◎ road net: lanes, road, highways, freeways/toll-roads, tunnels, bridges, ...
  - ◈ *Geotechnical:*
    - ◎ top layer soil composition
    - ◎ lower layer soil etc. composition, by depth levels
    - ◎ ground water occurrence, by depth levels
    - ◎ gas, oil occurrence, by depth levels

- ◈ *Meteorological:*
  - ◎ precipitation[12], for example, averaged by month (incl., perhaps, "hi/lo"), and possibly also changes by year, past and future
  - ◎ air humidity, by level, for example, averaged by month (incl., perhaps "hi/lo"), and possibly also changes by year, past and future
  - ◎ evaporation, by level , for example, averaged by month (incl., perhaps, "hi/lo"), and possibly also changes by year, past and future

[12] Precipitation: the amount of rain, snow, hail, etc., that has fallen at a given place within a given period, usually expressed in inches or centimeters of water.

- *Social and Citizen Economics:*
  - income distribution,
    currently, by year, ...
    and possibly also changes by year, past and future
  - housing situation,
    by housing category: apt., etc.; currently, by year, ...
    and possibly also changes by year, past and future
  - migration,
    and possibly also changes by year, past and future
  - social welfare support,
    by citizen category
    and possibly also changes by year, past and future
  - health status,
    by citizen category
    and possibly also changes by year, past and future
  - etcetera.

- *Industry and Business Economics:*
  - ... ,
  - ... ,
  - ... ,

  etcetera.
- *Etcetera.*

10.2.2. General on Attributes

- Parts (like TUS, GeoD, GeoT, ...) "possess" attributes.
- Attributes are intrisically associated with parts, that is, with a part type.
- All parts of a given type have the same attributes.
- We must distinguish between an *attribute name* and an *attribute value*.
  - Let $\eta\mathsf{A}_1$, $\eta\mathsf{A}_2$, ..., $\eta\mathsf{A}_n$ be all the attribute names of parts of type P.
  - Then two different parts, p_i and p_j, of type P,
    - may have the same value, $\mathsf{attr\_}\eta\mathsf{A}_k(\mathsf{p}_i)$ respectively $\mathsf{attr\_}\eta\mathsf{A}_k(\mathsf{p}_j)$, for attribute A_k,
    - or may have different values.
- If you try "remove" (whatever that would mean) an attribute
  - from a part, of a given type, say P,
  - then that 'part' is no longer of type P.

10.2.3. Urban Space Attributes – Formal

10.2.3.1 General

- Informal attribute names were given on slides 107–110 in the ⟡ itemized entries.
- We now treat attribute names and value abstractly.

71 Let $\eta A_1, \eta A_2, ..., \eta A_n$ be the (undoubtedly large) set of all attribute *names of interest* for some urban space.

72 And let $A_1, A_2, ..., A_n$ be type names for for corresponding attribute value sets.

73 The observation, from a part of type P, (which has attributes of name ηA) of values of type A is expressed by the attribute observer function $\mathsf{attr\_}\eta\mathsf{A}$.

type
71 $\eta\mathsf{A}_1,\eta\mathsf{A}_2,...,\eta\mathsf{A}_n$
72 $\mathsf{A}_1,\mathsf{A}_2,...,\mathsf{A}_n$
value
73 $\mathsf{attr\_}\eta\mathsf{A}_i$: P → A_i [for $1 \leq i \leq n$]

10.2.3.2 Structured Attributes

10.2.3.3 An Analysis of Structured Attributes

10.2.3.4 Structured Attribute Names

74
75
76
77
78

74
75
76
77
78

10.2.3.5 Structured Attribute Values

79
80
81
82
83

79
80
81
82
83