Urban Planning
Processes, Documents, Management
Shanghai TongJi Lectures

Dines Bjørner
Technical University of Denmark

September 11, 2017: 00:28 and September 11, 9:30–11:15, 2017

Slides:

- **My home page: .../ = http://www.imm.dtu.dk/~dibj/**
- **Report: .../2017/up/urban-planning.pdf**
- **Lecture 1 Slides: .../2017/up/tj-s1.pdf**
- **Lecture 2 Slides: .../2017/up/tj-s2.pdf**
- **Lecture 3 Slides: .../2017/up/tj-s3.pdf**
- **Lecture 4 Slides: .../2017/up/tj-s4.pdf**
- **Slides, 4/1, for all lectures: .../2017/up/tj-s-all.pdf**

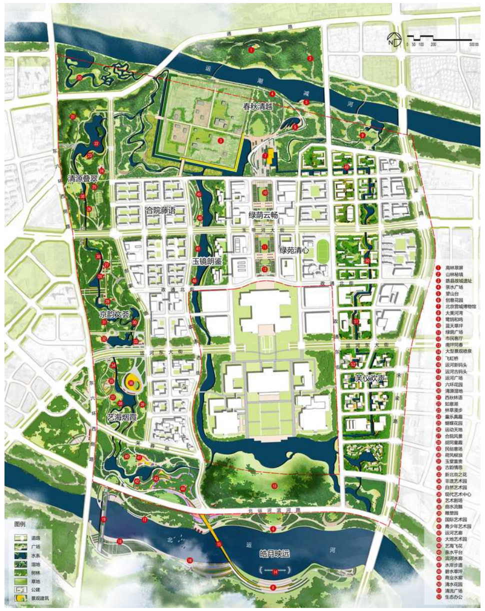
Four Lectures

First Lecture

- **Agenda:**

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

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

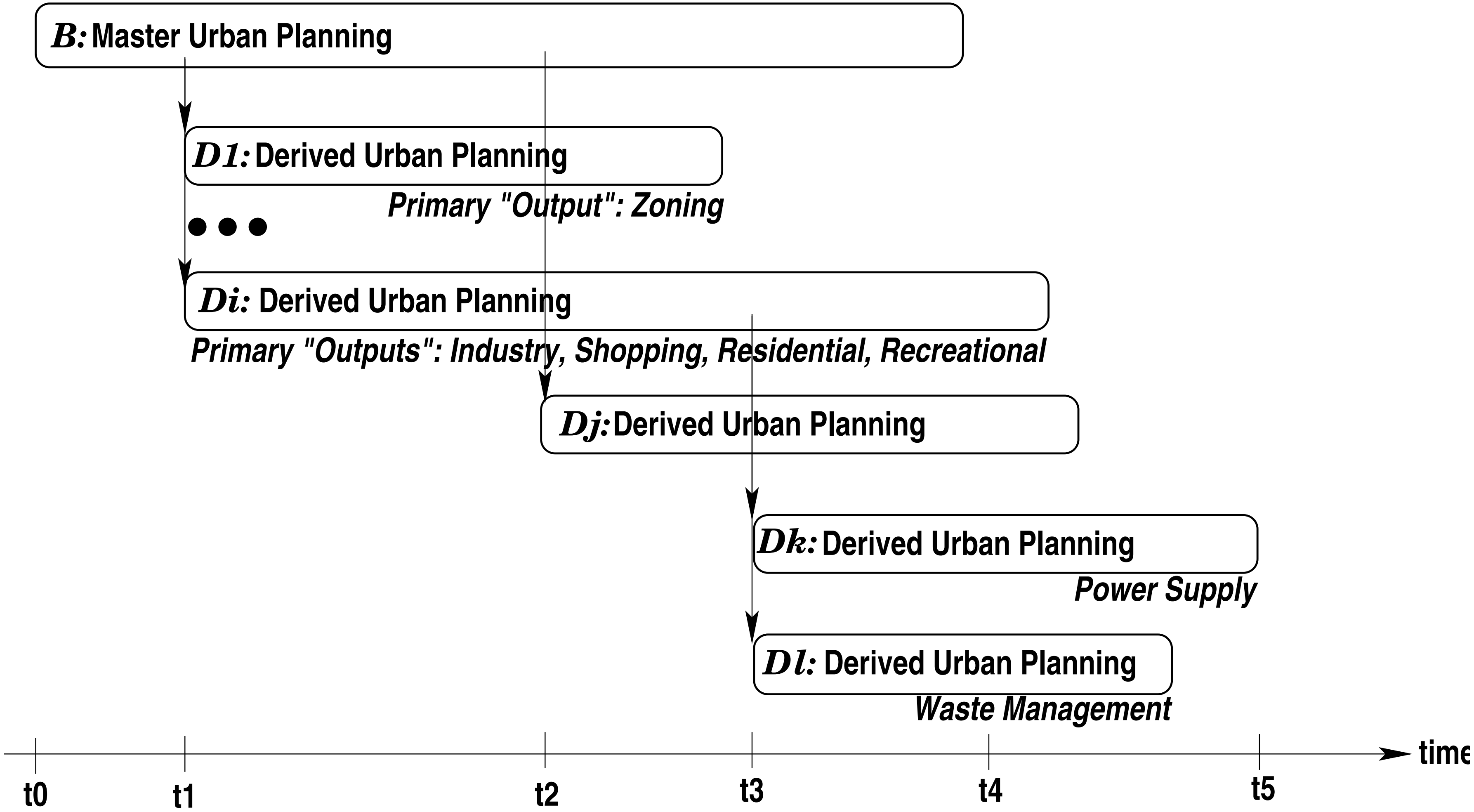
1.1.2.2.3. Secondary Plans

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

1.1.2.2.4. Infrastructure, Social &c. Plans

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

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
  X, Y, ..., Z

  we mean that X, Y, ..., Z are sets of further undefined values.

- When we write

  type
  $P = Q \times R$
  $S = W\text{-}\mathbf{set}$
  $F = A \to B \; (A \stackrel{\sim}{\to} B)$

  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** $X \rightarrow Y$,

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:

$$\mathsf{f(x) = y}$$

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 $\to$ Intg
square(i) $\equiv$ i$*$i

cube: Intg $\to$ Intg
cube(i) $\equiv$ i$*$i$*$i

addition: Intg $\times$ Intg $\to$ Intg
addition(i,j) $\equiv$ i+j

division: Intg $\times$ Intg $\stackrel{\sim}{\to}$ **Real**
division(i,j) $\equiv$ i/j **pre** j$\neq$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 $\rightarrow$ **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**

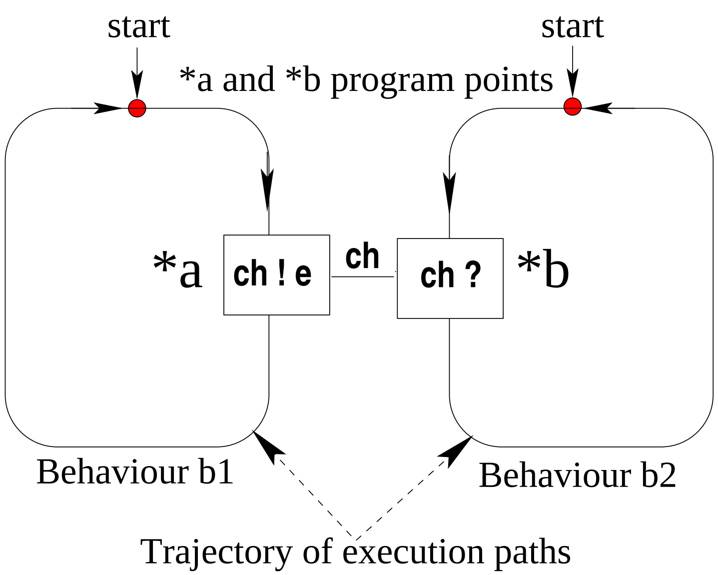
1.2.5.3 Behaviour Definitions, I

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.

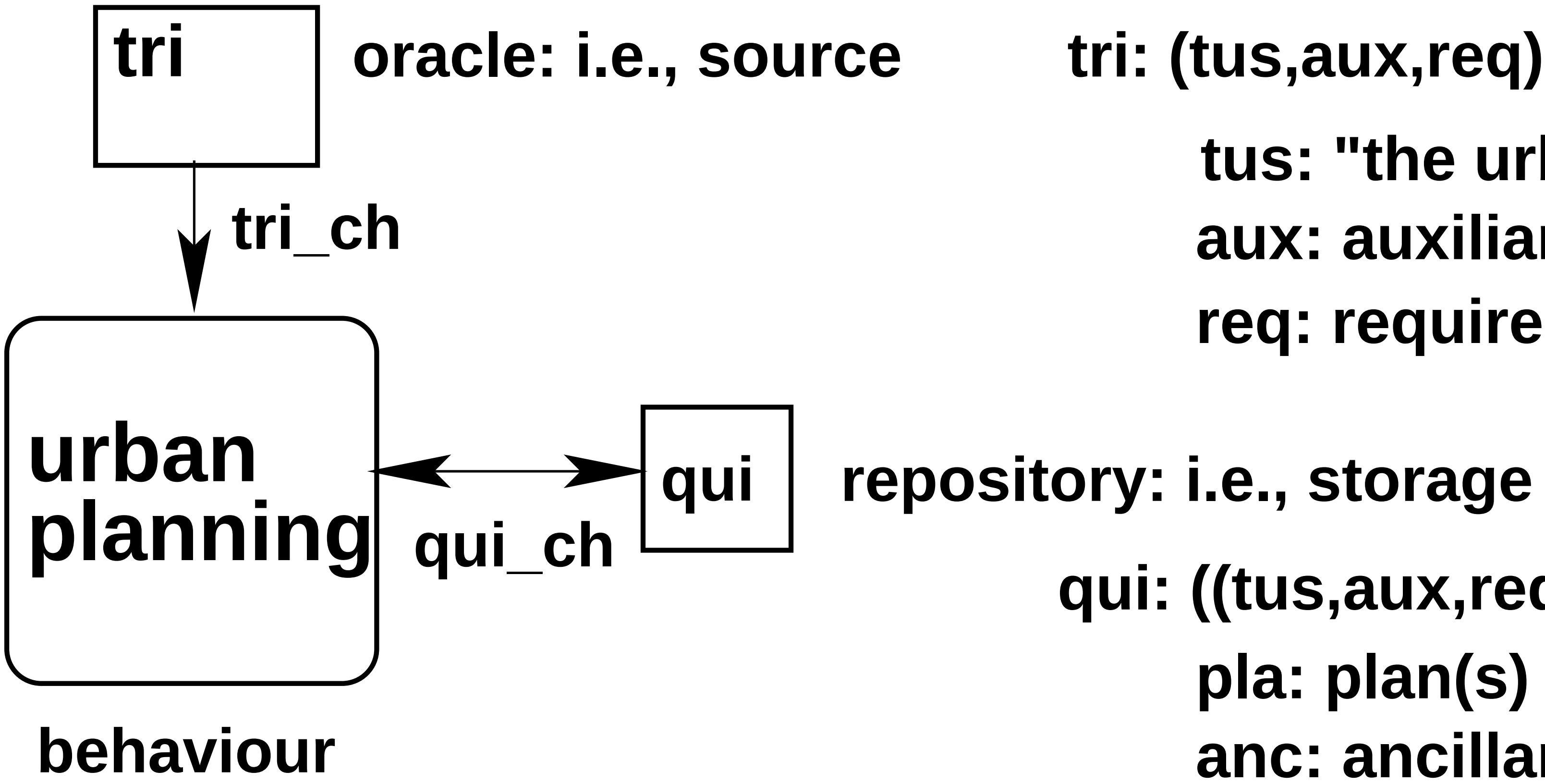
1.2.5.5 Behaviour Signatures, II

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"



tri: (tus,aux,req)

tus: "the urban space"

aux: auxiliary info

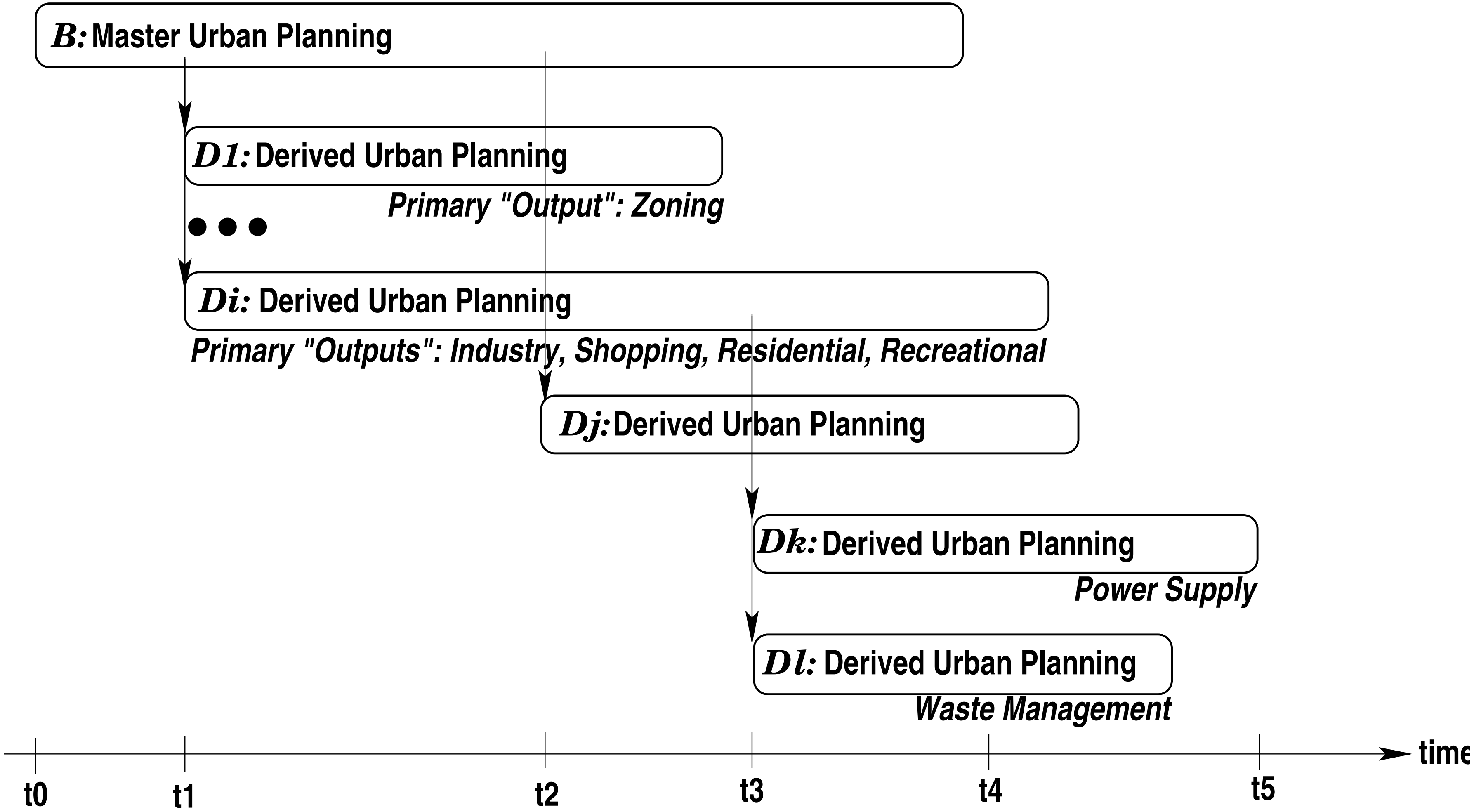
req: requirements

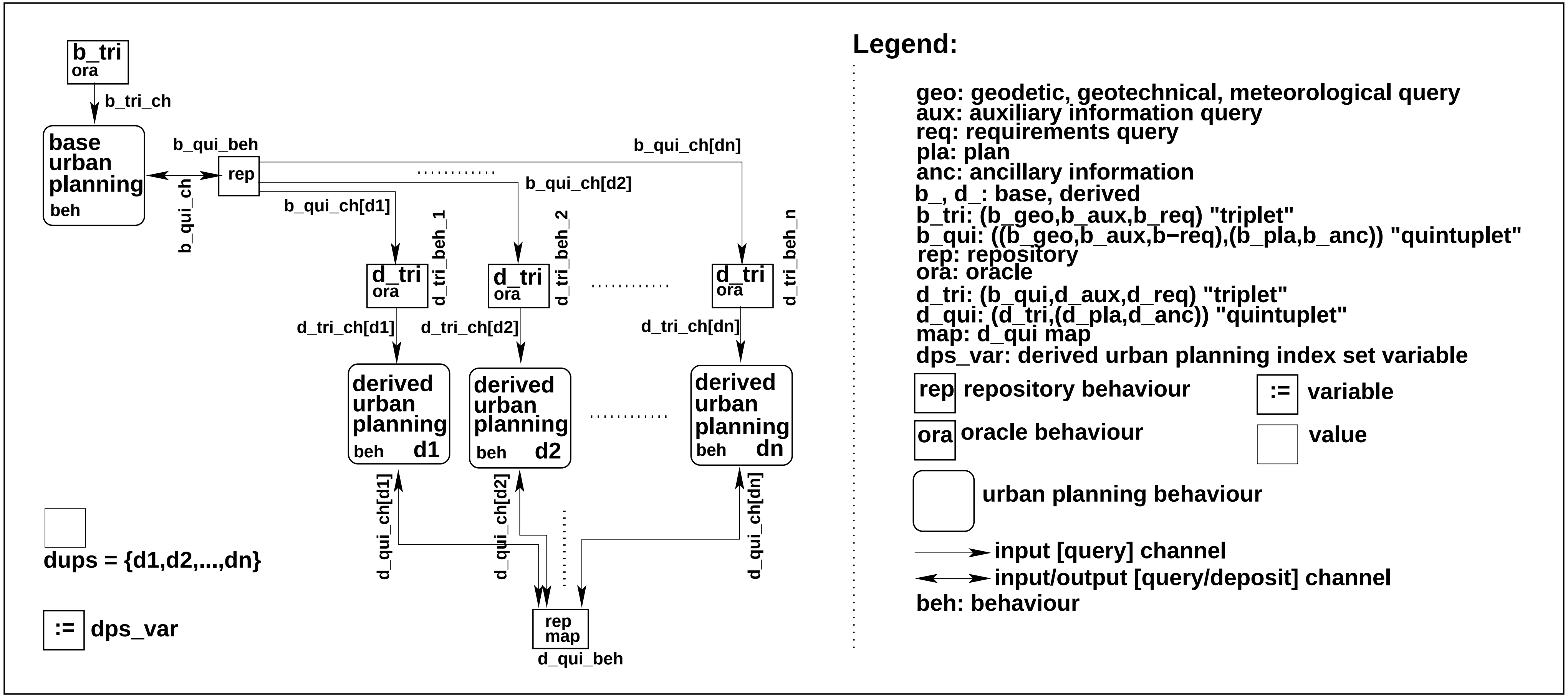
qui: ((tus,aux,req),(pla,anc))

pla: plan(s)

anc: ancillary info

1.3.2. “The Master and Derived Urban Plannings”




b_tri
ora
b_tri_ch
base urban planning beh
b_qui_beh
rep
b_qui_ch
b_qui_ch[d1]
b_qui_ch[d2]
b_qui_ch[dn]
d_tri ora
d_tri_beh_1
d_tri_beh_2
d_tri_beh_n
d_tri_ch[d1]
d_tri_ch[d2]
d_tri_ch[dn]
derived urban planning beh d1
derived urban planning beh d2
derived urban planning beh dn
d_qui_ch[d1]
d_qui_ch[d2]
d_qui_ch[dn]
rep map
d_qui_beh
dups = {d1,d2,...,dn}
:= dps_var
Legend:
geo: geodetic, geotechnical, meteorological query
aux: auxiliary information query
req: requirements query
pla: plan
anc: ancillary information
b_, d_: base, derived
b_tri: (b_geo,b_aux,b_req) "triplet"
b_qui: ((b_geo,b_aux,b−req),(b_pla,b_anc)) "quintuplet"
rep: repository
ora: oracle
d_tri: (b_qui,d_aux,d_req) "triplet"
d_qui: (d_tri,(d_pla,d_anc)) "quintuplet"
map: d_qui map
dps_var: derived urban planning index set variable
rep repository behaviour
:= variable
ora oracle behaviour
value
urban planning behaviour
input [query] channel
input/output [query/deposit] channel
beh: behaviour

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 $\rightarrow$ QUI $\rightarrow$ QUI

15 up_fct(tri)(qui) **as** (tri′,(pla,anc))

16 tri = tri′ $\wedge$

17 $\mathcal{P}_{\mathsf{base}}$(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
base_up_beh_0: Unit → in tri_ch out qui_ch Unit
base_up_beh_0() ≡
    let (tri,qui) = (tri_ch?,qui_ch?) in
    let qui = base_up_fct(tri)(qui) in
    qui_ch ! qui end end ;
    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" !

1.4.6. But is Necessary for Requirements Engineering and Software Design

- 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