Dines Bjørner

Fredsvej 11, DK-2840 Holte, Danmark

E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Pipelines*

October 30, 2009: 13:25

A Technical Note:
Work in progress.

*This technical note was edited while at The University of Edinburgh.

Contents

1

Pipeline Systems

s2

Fig. 1.1. The Planned Nabucco Pipeline: http://en.wikipedia.org/wiki/Nabucco_Pipeline

s3

- Named after Verdi's opera
- Gas pipeline
- 3300 kms
- 2011–2014, first gas flow: 2014; 2017–2019, more pipes
- 8 billion Euros
- Max flow: 31 bcmy: billion cubic meters a year
- **http://www.nabucco-pipeline.com/**

s4
s5

Fig. 1.2. The Planned Nabucco Pipeline: http://en.wikipedia.org/wiki/Nabucco_Pipeline

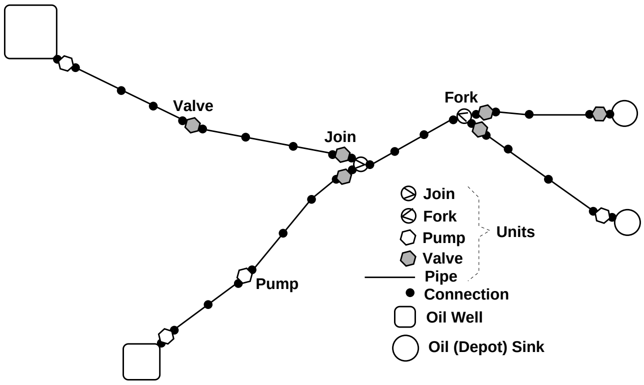

Fig. 1.3. An oil pipeline system

2

Non-Temporal Aspects of Pipelines

s6

Nets and Units: wells, pumps, pipes, valves, joins, forks and sinks. Net and unit attributes. Units states, but not state changes. We omit, in earlier chapters, consideration of "pigs" and "pig"-insertion and "pig"-extraction units.

2.1 Nets of Pipes, Valves, Pumps, Forks and Joins s7

1. We focus on nets, $n : N$, of pipes, $\pi : \Pi$, valves, $v : V$, pumps, $p : P$, forks, $f : F$, joins, $j : J$, wells, $w : W$ and sinks, $s : S$.
2. Units, $u : U$, are either pipes, valves, pumps, forks, joins, wells or sinks.
3. Units are explained in terms of disjoint types of PIpes, VAlves, PUmps, FOrks, JOins, WElls and SKs.[1]

type
  1 N, PI, VA, PU, FO, JO, WE, SK
  2 U = Π | V | P | F | J | S| W
  2 Π == mkΠ(pi:PI)
  2 V == mkV(va:VA)
  2 P == mkP(pu:PU)
  2 F == mkF(fo:FO)
  2 J == mkJ(jo:JO)
  2 W == mkW(we:WE)
  2 S == mkS(sk:SK)

2.2 Unit Identifiers and Unit Type Predicates s8

4. We associate with each unit a unique identifier, $ui : UI$.
5. From a unit we can observe its unique identifier.
6. From a unit we can observe whether it is a pipe, a valve, a pump, a fork, a join, a well or a sink unit.

type
  4 UI
value
  5 obs_UI: U → UI

[1] This is a mere specification language technicality.

6 is_*Π*: U → **Bool**, is_V: U → **Bool**, ..., is_J: U → **Bool**
is_*Π*(u) ≡ **case** u **of** mkPI(_) → **true**, _ → **false end**
is_V(u) ≡ **case** u **of** mkV(_) → **true**, _ → **false end**
...
is_S(u) ≡ **case** u **of** mkS(_) → **true**, _ → **false end**

2.3 Unit Connections s9

A connection is a means of juxtaposing units. A connection may connect two units in which case one can observe the identity of connected units from "the other side".

7. With a pipe, a valve and a pump we associate exactly one input and one output connection.
8. With a fork we associate a maximum number of output connections, m, larger than one.
9. With a join we associate a maximum number of input connections, m, larger than one.
10. With a well we associate zero input connections and exactly one output connection.
11. With a sink we associate exactly one input connection and zero output connections.

s10

value
7 obs_InCs,obs_OutCs: *Π*|V|P → {|1:**Nat**|}
8 obs_inCs: F → {|1:**Nat**|}, obs_outCs: F → **Nat**
9 obs_inCs: J → **Nat**, obs_outCs: J → {|1:**Nat**|}
10 obs_inCs: W → {|0:**Nat**|}, obs_outCs: W → {|1:**Nat**|}
11 obs_inCs: S → {|1:**Nat**|}, obs_outCs: S → {|0:**Nat**|}
axiom
8 ∀ f:F • obs_outCs(f) ≥ 2
9 ∀ j:J • obs_inCs(j) ≥ 2

s11 If a pipe, valve or pump unit is input-connected [output-connected] to zero (other) units, then it means that the unit input [output] connector has been sealed. If a fork is input-connected to zero (other) units, then it means that the fork input connector has been sealed. If a fork is output-connected to n units less than the maximum fork-connectability, then it means that the unconnected fork outputs have been sealed. Similarly for joins: "the other way around".

2.4 Net Observers and Unit Connections s12

12. From a net one can observe all its units.
13. From a unit one can observe the the pairs of disjoint input and output units to which it is connected:
    a) Wells can be connected to zero or one output unit — a pump.
    b) Sinks can be connected to zero or one input unit — a pump or a valve.
    c) Pipes, valves and pumps can be connected to zero or one input units and to zero or one output units.
    d) Forks, f, can be connected to zero or one input unit and to zero or n, $2 \leq n \leq$obs_Cs(f) output units.
    e) Joins, j, can be connected to zero or n, $2 \leq n \leq$obs_Cs(j) input units and zero or one output units.

s13

value
12 obs_Us: N → U-**set**
13 obs_cUIs: U → UI-**set** × UI-**set**
wf_Conns: U → **Bool**

```
    wf_Conns(u) ≡
       let (iuis,ouis) = obs_cUIs(u) in iuis ∩ ouis = {} ∧
       case u of
13a  mkW(_) → card iuis ∈ {0} ∧ card ouis ∈ {0,1},
13b  mkS(_) → card iuis ∈ {0,1} ∧ card ouis ∈ {0},
13c  mkΠ(_) → card iuis ∈ {0,1} ∧ card ouis ∈ {0,1},
13c  mkV(_) → card iuis ∈ {0,1} ∧ card ouis ∈ {0,1},
13c  mkP(_) → card iuis ∈ {0,1} ∧ card ouis ∈ {0,1},
13d  mkF(_) → card iuis ∈ {0,1} ∧ card ouis ∈ {0}∪{2..obs_inCs(j)},
13e  mkJ(_) → card iuis ∈ {0}∪{2..obs_inCs(j)} ∧ card ouis ∈ {0,1}
       end end
```

2.5 Well-formed Nets, Actual Connections s14

14. The unit identifiers observed by the obs_cUIs observer must be identifiers of units of the net.

```
axiom
   14  ∀ n:N,u:U • u ∈ obs_Us(n) ⇒
   14    let (iuis,ouis) = obs_cUIs(u) in
   14    ∀ ui:UI • ui ∈ iuis ∪ ouis ⇒
   14      ∃ u′:U • u′ ∈ obs_Us(n) ∧ u′≠u ∧ obs_UI(u′)=ui end
```

2.6 Well-formed Nets, No Circular Nets s15

15. By a route we shall understand a sequence of units.
16. Units form routes of the net.

```
type
   15  R = UI^ω
value
   16  routes: N → R-infset
   16  routes(n) ≡
   16    let us = obs_Us(n) in
   16    let rs = {⟨u⟩|u:U•u ∈ us} ∪ {r⌢r′|r,r′:R• {r,r′}⊆rs∧adj(r,r′)} in
   16    rs end end
```

s16

17. A route of length two or more can be decomposed into two routes
18. such that the least unit of the first route “connects” to the first unit of the second route.

```
value
   17   adj: R × R → Bool
   17   adj(fr,lr) ≡
   17     let (lu,fu)=(fr(len fr),hd lr) in
   18     let (lui,fui)=(obs_UI(lu),obs_UI(fu)) in
   18     let ((_,luis),(fuis,_))=(obs_cUIs(lu),obs_cUIs(fu)) in
   18     lui ∈ fuis ∧ fui ∈ luis end end end
```

19. No route must be circular, that is, the net must be acyclic.

```
value
  19  acyclic: N → Bool
  19    let rs = routes(n) in
  19    ∼∃ r:R•r ∈ rs⇒∃ i,j:Nat•{i,j}⊆inds r∧i≠j∧r(i)=r(j) end
```

2.7 Well-formed Nets, Special Pairs, wfN_SP s17

20. We define a "special-pairs" well-formedness function.
    a) Fork outputs are output-connected to valves.
    b) Join inputs are input-connected to valves.
    c) Wells are output-connected to pumps.
    d) Sinks are input-connected to either pumps or valves.

s18

```
value
  20   wfN_SP: N → Bool
  20   wfN_SP(n) ≡
  20     ∀ r:R • r ∈ routes(n) in
  20       ∀ i:Nat • {i,i+1}⊆inds r ⇒
  20         case r(i) of ∧
  20a          mkF(_) → ∀ u:U•adj(⟨r(i)⟩,⟨u⟩) ⇒ is_V(u),_→true end ∧
  20         case r(i+1) of
  20b          mkJ(_) → ∀ u:U•adj(⟨u⟩,⟨r(i)⟩) ⇒ is_V(u),_→true end ∧
  20         case r(1) of
  20c          mkW(_) → is_P(r(2)),_→true end ∧
  20         case r(len r) of
  20d          mkS(_) → is_P(r(len r−1))∨is_V(r(len r−1)),_→true end
```

The **true** clauses may be negated by other **case** distinctions' is_V or is_V clauses.

2.8 Special Routes, I s19

21. A pump-pump route is a route of length two or more whose first and last units are pumps and whose intermediate units are pipes or forks or joins.
22. A simple pump-pump route is a pump-pump route with no forks and joins.
23. A pump-valve route is a route of length two or more whose first unit is a pump, whose last unit is a valve and whose intermediate units are pipes or forks or joins.
24. A simple pump-valve route is a pump-valve route with no forks and joins.
25. A valve-pump route is a route of length two or more whose first unit is a valve, whose last unit is a pump and whose intermediate units are pipes or forks or joins.
26. A simple valve-pump route is a valve-pump route with no forks and joins.
27. A valve-valve route is a route of length two or more whose first and last units are valves and whose intermediate units are pipes or forks or joins.
28. A simple valve-valve route is a valve-valve route with no forks and joins.

s20

```
value
  21-28  ppr,sppr,pvr,spvr,vpr,svpr,vvr,svvr: R → Bool
           pre {ppr,sppr,pvr,spvr,vpr,svpr,vvr,svvr}(n): len n≥2

  21  ppr(r:⟨fu⟩⁀ℓ⁀⟨lu⟩) ≡ is_P(fu) ∧ is_P(lu) ∧ is_πfjr(ℓ)
  22  sppr(r:⟨fu⟩⁀ℓ⁀⟨lu⟩) ≡ ppr(r) ∧ is_πr(ℓ)
```

23 $\mathrm{pvr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{is\_P}(\mathrm{fu}) \wedge \mathrm{is\_V}(\mathrm{r}(\textbf{len}\ \mathrm{r})) \wedge \mathrm{is\_}\pi\mathrm{fjr}(\ell)$
24 $\mathrm{sppr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{ppr}(\mathrm{r}) \wedge \mathrm{is\_}\pi\mathrm{r}(\ell)$
25 $\mathrm{vpr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{is\_V}(\mathrm{fu}) \wedge \mathrm{is\_P}(\mathrm{lu}) \wedge \mathrm{is\_}\pi\mathrm{fjr}(\ell)$
26 $\mathrm{sppr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{ppr}(\mathrm{r}) \wedge \mathrm{is\_}\pi\mathrm{r}(\ell)$
27 $\mathrm{vvr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{is\_V}(\mathrm{fu}) \wedge \mathrm{is\_V}(\mathrm{lu}) \wedge \mathrm{is\_}\pi\mathrm{fjr}(\ell)$
28 $\mathrm{sppr}(\mathrm{r}:\langle \mathrm{fu}\rangle^\frown\ell^\frown\langle \mathrm{lu}\rangle) \equiv \mathrm{ppr}(\mathrm{r}) \wedge \mathrm{is\_}\pi\mathrm{r}(\ell)$

$\mathrm{is\_}\pi\mathrm{fjr},\mathrm{is\_}\pi\mathrm{r}: \mathrm{R} \to$ **Bool**
$\mathrm{is\_}\pi\mathrm{fjr}(\mathrm{r}) \equiv \forall\ \mathrm{u}:\mathrm{U}\bullet\mathrm{u} \in \textbf{elems}\ \mathrm{r} \Rightarrow \mathrm{is\_}\mathit{\Pi}(\mathrm{u}) \vee \mathrm{is\_F}(\mathrm{u}) \vee \mathrm{is\_J}(\mathrm{u})$
$\mathrm{is\_}\pi\mathrm{r}(\mathrm{r}) \equiv \forall\ \mathrm{u}:\mathrm{U}\bullet\mathrm{u} \in \textbf{elems}\ \mathrm{r} \Rightarrow \mathrm{is\_}\mathit{\Pi}(\mathrm{u})$

2.9 Special Routes, II s21

Given a unit of a route,
29. if they exist ($\exists$),
30. find the nearest pump or valve unit,
31. “upstream” and
32. “downstream” from the given unit.

s22

value
29 $\exists\mathrm{UpPoV}: \mathrm{U} \times \mathrm{R} \to$ **Bool**
29 $\exists\mathrm{DoPoV}: \mathrm{U} \times \mathrm{R} \to$ **Bool**
31 $\mathrm{find\_UpPoV}: \mathrm{U} \times \mathrm{R} \xrightarrow{\sim} (\mathrm{P}|\mathrm{V})$, **pre** $\mathrm{find\_UpPoV}(\mathrm{u,r}): \exists\mathrm{UpPoV}(\mathrm{u,r})$
32 $\mathrm{find\_DoPoV}: \mathrm{U} \times \mathrm{R} \xrightarrow{\sim} (\mathrm{P}|\mathrm{V})$, **pre** $\mathrm{find\_DoPoV}(\mathrm{u,r}): \exists\mathrm{DoPoV}(\mathrm{u,r})$
29 $\exists\mathrm{UpPoV}(\mathrm{u,r}) \equiv$
29 $\quad\exists\ \mathrm{i,j}\ \textbf{Nat}\bullet\{\mathrm{i,j}\}\subseteq\textbf{inds}\ \mathrm{r}\wedge\mathrm{i}\leq\mathrm{j}\wedge\{\mathrm{is\_V}|\mathrm{is\_P}\}(\mathrm{r(i)})\wedge\mathrm{u}=\mathrm{r(j)}$
29 $\exists\mathrm{DoPoV}(\mathrm{u,r}) \equiv$
29 $\quad\exists\ \mathrm{i,j}\ \textbf{Nat}\bullet\{\mathrm{i,j}\}\subseteq\textbf{inds}\ \mathrm{r}\wedge\mathrm{i}\leq\mathrm{j}\wedge\mathrm{u}=\mathrm{r(i)}\wedge\{\mathrm{is\_V}|\mathrm{is\_P}\}(\mathrm{r(j)})$
31 $\mathrm{find\_UpPoV}(\mathrm{u,r}) \equiv$
31 $\quad$**let** $\mathrm{i,j}:\textbf{Nat}\bullet\{\mathrm{i,j}\}\subseteq\mathrm{indsr}\wedge\mathrm{i}\leq\mathrm{j}\wedge\{\mathrm{is\_V}|\mathrm{is\_P}\}(\mathrm{r(i)})\wedge\mathrm{u}=\mathrm{r(j)}$ **in** $\mathrm{r(i)}$ **end**
32 $\mathrm{find\_DoPoV}(\mathrm{u,r}) \equiv$
32 $\quad$**let** $\mathrm{i,j}:\textbf{Nat}\bullet\{\mathrm{i,j}\}\subseteq\mathrm{indsr}\wedge\mathrm{i}\leq\mathrm{j}\wedge\mathrm{u}=\mathrm{r(i)}\wedge\{\mathrm{is\_V}|\mathrm{is\_P}\}(\mathrm{r(j)})$ **in** $\mathrm{r(j)}$ **end**

3

State Attributes of Pipeline Units

s23

By a state attribute of a unit we mean either of the following three kinds: (i) the open/close states of valves and the pumping/not_pumping states of pumps; (ii) the maximum (laminar) oil flow characteristics of all units; and (iii) the current oil flow and current oil leak states of all units. s24

33. Oil flow, $\phi : \Phi$, is measured in volume per time unit.
34. Pumps are either pumping or not pumping, and if not pumping they are closed.
35. Valves are either open or closed.
36. Any unit permits a maximum input flow of oil while maintaining laminar flow. We shall assume that we need not be concerned with turbulent flows.
37. At any time any unit is sustaining a current input flow of oil (at its input(s)).
38. While sustaining (even a zero) current input flow of oil a unit leaks a current amount of oil (within the unit).

s25

```
type
   33  Φ
   34  PΣ == pumping | not_pumping
   34  VΣ == open | closed
value
           −,+: Φ × Φ → Φ, <,=,>: Φ × Φ → Bool
   34      obs_PΣ: P → PΣ
   35      obs_VΣ: V → VΣ
   36–38  obs_LamiΦ.obs_CurrΦ,obs_LeakΦ: U → Φ
   is_Open: U → Bool
      case u of
         mkΠ(_)→true,mkF(_)→true,mkJ(_)→true,mkW(_)→true,mkS(_)→true,
         mkP(_)→obs_PΣ(u)=pumping,
         mkV(_)→obs_VΣ(u)=open
      end
   acceptable_LeakΦ, excessive_LeakΦ: U → Φ
axiom
   ∀ u:U • excess_LeakΦ(u) > accept_LeakΦ(u)
```

3.1 Flow Laws

s26

The sum of the current flows into a unit equals the the sum of the current flows out of a unit minus the (current) leak of that unit. This is the same as the current flows out of a unit equals the current flows into a unit minus the (current) leak of that unit. The above represents an interpretation which justifies the below laws. s27

39. When, in Item 37, for a unit u, we say that at any time any unit is sustaining a current input flow of oil, and when we model that by obs_CurrΦ(u) then we mean that obs_CurrΦ(u) - obs_LeakΦ(u) represents the flow of oil from its outputs.

value

39 obs_inΦ: U $\to$ Φ
39 obs_inΦ(u) $\equiv$ obs_CurrΦ(u)
39 obs_outΦ: U $\to$ Φ

law:

39 $\forall$ u:U • obs_outΦ(u) = obs_CurrΦ(u)$-$obs_LeakΦ(u)

s28

40. Two connected units enjoy the following flow relation:
    a) If
       i. two pipes, or
       ii. a pipe and a valve, or
       iii. a valve and a pipe, or
       iv. a valve and a valve, or
       v. a pipe and a pump, or
       vi. a pump and a pipe, or
       vii. a pump and a pump, or
       viii. a pump and a valve, or
       ix. a valve and a pump

       are immediately connected
    b) then
       i. the current flow out of the first unit's connection to the second unit
       ii. equals the current flow into the second unit's connection to the first unit

law:

40a $\forall$ u,u$'$:U • {is_Π,is_V,is_P,is_W}(u$'$|u$''$) $\wedge$ adj($\langle$u$\rangle$,$\langle$u$'\rangle$)
40a is_Π(u)$\vee$is_V(u)$\vee$is_P(u)$\vee$is_W(u) $\wedge$
40a is_Π(u$'$)$\vee$is_V(u$'$)$\vee$is_P(u$'$)$\vee$is_S(u$'$)
40b $\Rightarrow$ obs_outΦ(u)=obs_inΦ(u$'$)

s29 A similar law can be established for forks and joins. For a fork output-connected to, for example, pipes, valves and pumps, it is the case that for each fork output the out-flow equals the in-flow for that output-connected unit. For a join input-connected to, for example, pipes, valves and pumps, it is the case that for each join input the in-flow equals the out-flow for that input-connected unit. We leave the formalisation as an exercise.

3.2 Possibly Desirable Properties s30

41. Let r be a route of length two or more, whose first unit is a pump, p, whose last unit is a valve, v and whose intermediate units are all pipes: if the pump, p is pumping, then we expect the valve, v, to be open.
42. Let r be a route of length two or more, whose first unit is a pump, p, whose last unit is another pump, p' and whose intermediate units are all pipes: if the pump, p is pumping, then we expect pump p'', to also be pumping.
43. Let r be a route of length two or more, whose first unit is a valve, v, whose last unit is a pump, p and whose intermediate units are all pipes: if the valve, v is closed, then we expect pump p, to not be pumping.
44. Let r be a route of length two or more, whose first unit is a valve, v', whose last unit is a valve, v'' and whose intermediate units are all pipes: if the valve, v' is in some state, then we expect valve v'', to also be in the same state.

s31
s32

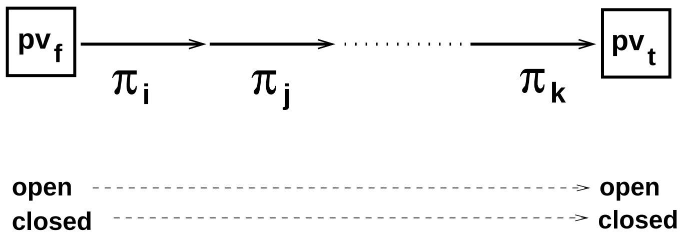

Fig. 3.1. pv: Pump or valve, π: pipe

desirable properties:

41 $\forall$ r:R • spvr(r) $\wedge$
41 **spvr_prop(r):** obs_PΣ(**hd** r)=pumping $\Rightarrow$ obs_PΣ(r(**len** r))=open

42 $\forall$ r:R • sppr(r) $\wedge$
42 **sppr_prop(r):** obs_PΣ(**hd** r)=pumping$\Rightarrow$obs_PΣ(r(**len** r))=pumping

43 $\forall$ r:R • svpr(r) $\wedge$
43 **svpr_prop(r):** obs_PΣ(**hd** r)=open$\Rightarrow$obs_PΣ(r(**len** r))=pumping

44 $\forall$ r:R • svvr(r) $\wedge$
44 **svvr_prop(r):** obs_PΣ(**hd** r)=obs_PΣ(r(**len** r))

4

Pipeline Actions

s33

4.1 Simple Pump and Valve Actions

45. Pumps may be set to pumping or reset to not pumping irrespective of the pump state.
46. Valves may be set to be open or to be closed irrespective of the valve state.
47. In setting or resetting a pump or a valve a desirable property may be lost.

```
value
  45 pump_to_pump, pump_to_not_pump: P → N → N
  46 valve_to_open, valve_to_close: V → N → N
```

s34

```
value
  45 pump_to_pump(p)(n) as n′
  45   pre p ∈ obs_Us(n)
  45   post let p′:P•obs_UI(p)=obs_UI(p′) in
  45          obs_PΣ(p′)=pumping∧else_equal(n,n′)(p,p′) end
  45 pump_to_not_pump(p)(n) as n′
  45   pre p ∈ obs_Us(n)
  45   post let p′:P•obs_UI(p)=obs_UI(p′) in
  45          obs_PΣ(p′)=not_pumping∧else_equal(n,n′)(p,p′) end
  46 valve_to_open(v)(n) as n′
  45   pre v ∈ obs_Us(n)
  46   post let v′:V•obs_UI(v)=obs_UI(v′) in
  45          obs_VΣ(v′)=open∧else_equal(n,n′)(v,v′) end
  46 valve_to_close(v)(n) as n′
  45   pre v ∈ obs_Us(n)
  46   post let v′:V•obs_UI(v)=obs_UI(v′) in
  45          obs_VΣ(v′)=close∧else_equal(n,n′)(v,v′) end
```

s35

```
value
  else_equal: (N×N) → (U×U) → Bool
  else_equal(n,n′)(u,u′) ≡
     obs_UI(u)=obs_UI(u′)
   ∧ u ∈ obs_Us(n)∧u′ ∈ obs_Us(n′)
   ∧ omit_Σ(u)=omit_Σ(u′)
   ∧ obs_Us(n)\{u}=obs_Us(n)\{u′}
   ∧ ∀ u″:U•u″ ∈ obs_Us(n)\{u} ≡ u″ ∈ obs_Us(n′)\{u′}
```

$\text{omit\_}\Sigma$: U → $\text{U}_{\text{no\_state}}$ −−− "magic" function

=: $\text{U}_{\text{no\_state}} \times \text{U}_{\text{no\_state}}$ → **Bool**

axiom

∀ u,u′:U•$\text{omit\_}\Sigma$(u)=$\text{omit\_}\Sigma$(u′) ≡ obs_UI(u)=obs_UI(u′)

4.2 Events s36

4.2.1 Unit Handling Events

48. Let n be any acyclic net.
48. If there exists p, p', v, v', pairs of distinct pumps and distinct valves of the net,
48. and if there exists a route, r, of length two or more of the net such that
49. all units, u, of the route, except its first and last unit, are pipes, then
50. if the route "spans" between p and p' and the *simple desirable property*, sppr(r), does not hold for the route, then we have a possibly undesirable event — that occurred as soon as sppr(r) did not hold;
51. if the route "spans" between p and v and the *simple desirable property*, spvr(r), does not hold for the route, then we have a possibly undesirable event;
52. if the route "spans" between v and p and the *simple desirable property*, svpr(r), does not hold for the route, then we have a possibly undesirable event; and
53. if the route "spans" between v and v' and the *simple desirable property*, svvr(r), does not hold for the route, then we have a possibly undesirable event.

s37

events:

48 ∀ n:N • acyclic(n) ∧
48 ∃ p,p′:P,v,v′:V • {p,p′,v,v′}⊆obs_Us(n)⇒
48 ∧ ∃ r:R • routes(n) ∧
49 ∀ u:U • u ∈ **elems**(r)\{**hd** r,r(**len** r)} ⇒ is_Π(i) ⇒
50 p=**hd** r∧p′=r(**len** r) ⇒ ∼sppr_prop(r) ∧
51 p=**hd** r∧v=r(**len** r) ⇒ ∼spvr_prop(r) ∧
52 v=**hd** r∧p=r(**len** r) ⇒ ∼svpr_prop(r) ∧
53 v=**hd** r∧v′=r(**len** r) ⇒ ∼svvr_prop(r)

4.2.2 Foreseeable Accident Events s38

A number of foreseeable accidents may occur.

54. A unit ceases to function, that is,
    a) a unit is clogged,
    b) a valve does not open or close,
    c) a pump does not pump or stop pumping.
55. A unit gives rise to excessive leakage.
56. A well becomes empty or a sunk becomes full.
57. A unit, or a connected net of units gets on fire.
58. Or a number of other such "accident".

s39

54
55
56
57
58

4.3 Well-formed Operational Nets s40

59. A well-formed operational net
60. is a well-formed net
    a) with at least one well, w, and at least one sink, s,
    b) and such that there is a route in the net between w and s.

value

59 wf_OpN: N → **Bool**
59 wf_OpN(n) ≡
60 satisfies axiom 14 on page 9 ∧ acyclic(n): Item 19 on page 9 ∧
60 wfN_SP(n): Item 20 on page 10 ∧
60 satisfies flow laws, 39 on page 14 and 40 on page 14 ∧
60a ∃ w:W,s:S • {w,s}⊆obs_Us(n) ⇒
60b ∃ r:R• ⟨w⟩⌢r⌢⟨s⟩ ∈ routes(n)

4.4 Orderly Action Sequences s41

4.4.1 Initial Operational Net

61. Let us assume a notion of an initial operational net.
62. Its pump and valve units are in the following states
    a) all pumps are not_pumping, and
    b) all valves are closed.

value

61 initial_OpN: N → **Bool**
62 initial_OpN(n) ≡ wf_OpN(n) ∧
62a ∀ p:P • p ∈ obs_Us(n) ⇒ obs_PΣ(p)=not_pumping ∧
62b ∀ v:V • v ∈ obs_Us(n) ⇒ obs_VΣ(p)=closed

4.4.2 Oil Pipeline Preparation and Engagement s42

63. We now wish to prepare a pipeline from some well, $w : W$, to some sink, $s : S$, for flow.
    a) We assume that the underlying net is operational wrt. w and s, that is, that there is a route, r, from w to s.
    b) Now, an orderly action sequence for engaging route r is to "work backwards", from s to w
    c) setting encountered pumps to pumping and valves to open.

In this way the system is well-formed wrt. the desirable sppr, spvr, svpr and svvr properties. Finally, setting the pump adjacent to the (preceding) well starts the system. s43

```
value
  63   prepare_and_engage: W × S → N ~→ N
  63   prepare_and_engage(w,s)(n) ≡
  63a    let r:R • ⟨w⟩⌢r⌢⟨s⟩ ∈ routes(n) in
  63b    action_sequence(⟨w⟩⌢r⌢⟨s⟩)(len⟨w⟩⌢r⌢⟨s⟩)(n) end
  63    pre ∃ r:R • ⟨w⟩⌢r⌢⟨s⟩ ∈ routes(n)

  63c  action_sequence: R → Nat → N → N
  63c  action_sequence(r)(i)(n) ≡
  63c    if i=1 then n else
  63c    case r(i) of
  63c      mkV(_) → action_sequence(r)(i−1)(valve_to_open(r(i))(n)),
  63c      mkP(_) → action_sequence(r)(i−1)(pump_to_pump(r(i))(n)),
  63c      _ → action_sequence(r)(i−1)(n)
  63c    end end
```

4.5 Emergency Actions s44

64. If a unit starts leaking excessive oil
    a) then nearest up-stream valve(s) must be closed,
    b) and any pumps in-between this (these) valves and the leaking unit must be set to not_pumping — following an orderly sequence.
65. If, as a result, for example, of the above remedial actions, any of the desirable properties cease to hold
    a) then — a ha !
    b) Left as an exercise.

5

Connectors

s45

The interface , that is, the possible “openings”, between adjacent units have not been explored. Likewise the for the possible “openings” of “begin” or “end” units, that is, units not having their input(s), respectively their “output(s)” connected to anything, but left “exposed” to the environment. We now introduce a notion of connectors: abstractly you may think of connectors as concepts, and concretely as “fittings” with bolts and nuts, or “weldings”, or “plates” inserted onto “begin” or “end” units. s46

66. There are connectors and connectors have unique connector identifiers.
67. From a connector one can observe its uniwue connector identifier.
68. From a net one can observe all its connectors
69. and hence one can extract all its connector identifiers.
70. From a connector one can observe a pair of “optional” (distinct) unit identifiers:
    a) An optional unit identifier is
    b) either a unit identifier of some unit of the net
    c) or a `‘‘nil’’` “identifier”.
71. In an observed pair of “optional” (distinct) unit identifiers
    - there can not be two `‘‘nil’’` “identifiers”.
    - or the possibly two unit identifiers must be distinct

s47

type
  66 K, KI
value
  67 obs_KI: $K \to KI$
  68 obs_Ks: $N \to K$-**set**
  69 xtr_KIS: $N \to KI$-**set**
  69 xtr_KIs(n) $\equiv \{obs\_KI(k)|k{:}K \bullet k \in obs\_Ks(n)\}$
type
  70 $oUIp' = (UI|\{|nil|\}) \times (UI|\{|nil|\})$
  70 $oUIp = \{|ouip{:}oUIp' \bullet wf\_oUIp(ouip)|\}$
value
  70 obs_oUIp: $K \to oUIp$
  71 wf_oUIp: $oUIp' \to$ **Bool**
  71 wf_oUIp(uon,uon$'$) $\equiv$
  71   $uon{=}nil \Rightarrow uon' {\neq} nil \lor uon'{=}nil \Rightarrow uon {\neq} nil \lor uon {\neq} uon'$

s48

72. Under the assumption that a fork unit cannot be adjacent to a join unit
73. we impose the constraint thet no two distinct connectors feature the same pair of actual (distinct) unit identifiers.

74. The first proper unit identifier of a pair of “optional” (distinct) unit identifiers must identify a unit of the net.
75. The second proper unit identifier of a pair of “optional” (distinct) unit identifiers must identify a unit of the net.

s49

```
axiom
   72  ∀ n:N,u,u′:U•{u.u′}⊆obs_Us(n)∧adj(u,u′)⇒ ∼(is_F(u)∧is_J(u′))

   73  ∀ k,k′:K•obs_KI(k)≠obs_KI(k′)⇒
           case (obs_oUIp(k),obs_oUIp(k′)) of
               ((nil,ui),(nil,ui′)) → ui≠ui′,
               ((nil,ui),(ui′,nil)) → false,
               ((ui,nil),(nil,ui′)) → false,
               ((ui,nil),(ui′,nil)) → ui≠ui′,
               _ → false
           end
```

s50

```
   ∀ n:N,k:K•k ∈ obs_Ks(n) ⇒
      case obs_oUIp(k) of
    74       (ui,nil) → ∃UI(ui)(n)
    75       (nil,ui) → ∃UI(ui)(n)
    74-75  (ui,ui′) → ∃UI(ui)(n)∧∃UI(ui′)(n)
      end
value
   ∃UI: UI → N → Bool
   ∃UI(ui)(n) ≡ ∃ u:U•u ∈ obs_Us(n)∧obs_UI(u)=ui
```

6

Temporal Aspects of Pipelines

s51

The else_qual(u,u′)(n,n′) function definition represents a gross simplification. It ignores the actual flow which changes as a result of setting alternate states, and hence the net state. We now wish to capture the dynamics of flow. We shall do so using the Duration Calculus — a continuous time, integral temporal logic that is semantically and proof system "integrated" with RSL:

> Zhou ChaoChen and Michael Reichhardt Hansen
> Duration Calculus: A Formal Approach to Real-time Systems
> Monographs in Theoretical Computer Science
> The EATCS Series
> Springer 2004

MORE TO COME

7

A CSP Model of Pipelines

s52

We recapitulate Sect. 5 — now adding connectors to our model:

76. From an oil pipeline system one can observe units and connectors.
77. Units are either well, or pipe, or pump, or valve, or join, or fork or sink units.
78. Units and connectors have unique identifiers.
79. From a connector one can observe the ordered pair of the identity of the two from-, respectively to-units that the connector connects.

s53

type
77 OPLS, U, K
79 UI, KI
value
77 obs_Us: OPLS → U-**set**, obs_Ks: OPLS → K-**set**
78 is_WeU, is_PiU, is_PuU, is_VaU, is_JoU, is_FoU, is_SiU: U → **Bool** [mutually exclusive]
79 obs_UI: U → UI, obs_KI: K → KI
80 obs_UIp: K → (UI|{nil}) × (UI|{nil})

s54

Above, we think of the types OPLS, U, K, UI and KI as denoting semantic entities. Below, in the next section, we shall consider exactly the same types as denoting syntactic entities ! s55

80. There is given an oil pipeline system, opls.
81. To every unit we associate a CSP behaviour.
82. Units are indexed by their unique unit identifiers.
83. To every connector we associate a CSP channel.
    Channels are indexed by their unique "k"onnector identifiers.
84. Unit behaviours are cyclic and over the state of their (static and dynamic) attributes, represented by u.
85. Channels, in this model, have no state.
86. Unit behaviours communicate with neighbouring units — those with which they are connected.
87. Unit functions, $\mathcal{U}_i$, change the unit state.
88. The pipeline system is now the parallel composition of all the unit behaviours.

s56

Editorial Remark: Our use of the term unit and the RSL literal **Unit** may seem confusing, and we apologise. The former, unit, is the generic name of a well, pipe, or pump, or valve, or join, or fork, or sink. The literal **Unit**, in a function signature, before the → "announces" that the function takes no argument.[1] The literal **Unit**, in a function signature, after the → "announces", as used here, that the function never terminates. s57

[1] **Unit** is a type name; () is the only value of type **Unit**.

```
value
81  opls:OPLS
channel
84  {ch[ ki ]|k:KI,k:K•k ∈ obs_Ks(opls)∧ki=obs_KI(k)} M
value
89  pipeline_system: Unit → Unit
89  pipeline_system() ≡
82    ‖ {unit(ui)(u)|u:U•u ∈ obs_Us(opls)∧ui=obs_UI(u)}

83  unit: ui:UI → U →
87        in,out {ch[ ki ]|k:K,ki:KI•k ∈ obs_Ks(opls)∧ki=obs_KI(k)∧
87                   let (ui′,ui″)=obs_UIp(k) in ui ∈{ui′,ui″}\{nil} end}  Unit
85  unit(ui)(u) ≡ let u′ = 𝒰ᵢ(ui)(u) in unit(ui)(u′) end

88  𝒰ᵢ: ui:UI → U →
88        in,out {ch[ ki ]|k:K,ki:KI•k ∈ obs_Ks(opls)∧ki=obs_KI(k)∧
88                   let (ui′,ui″)=obs_UIp(k) in ui ∈{ui′,ui″}\{nil} end}  U
```

$\boxed{\text{MORE TO COME}}$

8

A Language of Units and Connectors

s58

We extend our variety of pipeline system units with a "pig" insertion and extraction unit. A "pig" is a cylindrical devise that "just" pits within a pipe and can "travel", in the direction of oil flow, from one "pig" insertionunit to another "pig" extraction unit. "Pig" insertion and extraction units can only be infixed between pipe units. They have four connectors, two conventional s59
connectors that only connect to pipes: one from an input, the other to an output pipe. and two "pig" insertion and extraction "connectors" one at which to insert a "pig", the other at which to extract a "pig". At most one "pig" may be "travelling" along any number of otherwise connected pipe units.

8.1 Language Syntax s60

We refer to Fig. 8.1.

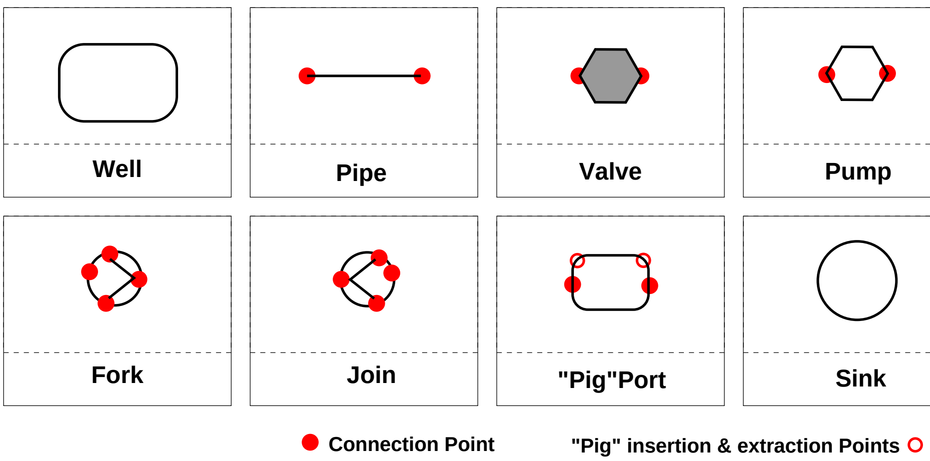

Fig. 8.1. A Diagrammatic Rendition of Pipeline System Units

s61

These units, including the "pig" insertion and extraction unit, and these connectors are syntactic entities; that is in constrast to the units and connectors of the model of Sects. 2–4 — they were semantic entities. Also the model U's, UI's, K's and KI's of Sect. 7 was "purely" syntactic.

s62

89. There are (syntactic renditions of) wells, pipes, valves, pumps, forks, joins, "pig" insertion and extraction and sink units and of connectors and "pig" insertion and extraction points.

90. These are the language units: well units, pipe units, valve units, pump units, fork units, join units, “pig” nsertion and extraction units and sink units.
91. A syntactic well unit has one syntactic pipewell and one syntactic output connector.
92. A syntactic pipe unit has one syntactic input connector, one syntactic pipe and one syntactic output connector.
93. A syntactic valve unit has one syntactic input connector, one syntactic valvr and one syntactic output connector.
94. A syntactic pump unit has one syntactic input connector, one syntactic pump and one syntactic output connector. s63
95. A syntactic fork unit has two or more named syntactic input connectors, one syntactic fork and one syntactic output connector.
96. A syntactic join unit has one syntactic input connector, one syntactic join and two or more named syntactic output connector.
97. A syntactic pig unit has one syntactic input connector, one syntactic “pig” insertion point, one syntactic “pig” insertion and extraction unit, one syntactic output connector and one syntactic “pig” extraction point.
98. A syntactic sink unit has one syntactic input connector and one syntactic sink.

s64

type
90 Well,Pipe,Valve,Pump,Fork,Join,PigIE,Sink,Co,PK
91 LU = WelU | PipU | VaU | PuU | FoU | JoU | PigU | SnkU
92 WelU == mkWeU(we:Well,oc:(ok:KI,oco:Co))
93 PipU == mkPiU(ic:(ik:KI,ico:Co),pi:Pipe,oc:(ok:KI,oco:Co))
94 VaU == mkVaU(ic:(ik:KI,ico:Co),v:Valve,oc:(ok:KI,oco:Co))
95 PuU == mkPuU(ic:(ik:KI,ico:Co),pu:Pump,oc:(ok:KI,oco:Co))
96 FoU == mkFoU(ic:(ik:KI,ico:Co),f:Fork,oc:(KI $\overrightarrow{m}$ Co))
97 JoU == mkJoU(ic:(KI $\overrightarrow{m}$ Co),j:Join,oc:(ok:KI,oco:Co))
98 PigU == mkVaU(i:(ic:Co,ipc:PK),pigie:PigIE,o:(oc:C,opc:PK))
99 SnkU == mkSnU(i:(ik:KI,ico:Co),sn:Sink)

s65

99. A syntactic pipeline system (specification) is a composition of one or more units,
100. where syntactic non-pipe units are composed with syntactic pipe units such that the connection points “agree”.
101. For all language units,
102. u let us identify all
   a) input connectors, ikis, (one or more), and
   b) output connectors, okis, (one or more). s66
101. “Agreement” is now defined; for all language units:
   a) For all input connector identifiers, ki,
   b) there exists a unique and different unit, u′, such that ki is in its set of output connector identifiers.
103. and — vice versa:
   a) for all output connector identifiers, ki,
   b) there exists a unique and different unit, u′, such that ki is in its set of input connector identifiers.

s67

type
100 SPLS′ = LU-**set**
100 SPLS = {| spls:SPLS′ • **card** spls≥1 ∧ wf_SPLS(spls) |}
value
101 wf_SPLS: SPLS′ → **Bool**
101 wf_SPLS(spls) ≡
102 ∀ u:LU • u ∈ spls ⇒

```
103        let (ikis,okis) = identify_KIs(u) in
103a         ∀ ki:KI•ki ∈ ikis ⇒
103b            ∃! u′:LU • u≠u′ ⇒
103b              let  (_,okis′) = identify_KIs(u′) in ki ∈ okis′ end
104        ∧
104a         ∀ ki:KI•ki ∈ okis ⇒
104b            ∃! u′:LU •  u≠u′ ⇒
104b              let  (ikis′,_) = identify_KIs(u′) in ki ∈ ikis′ end end
```

s68

```
value
  103 identify_KIs: U → KI-set×KI-set
  103 identify_KIs(u) ≡
  103     case u of
  103         mkWelU(_,_,(ok,_)) → ({},{ok}),
  103         mkPipU((ik,_),_,(ok,_)) → ({ik},{ok}),
  103         mkVaU((ik,_),_,(ok,_)) → ({ik},{ok}),
  103         mkPuU((ik,_),_,(ok,_)) → ({ik},{ok}),
  103         mkFoU((ik,_),_,okim) → ({ik},dom okim),
  103         mkJoU(ikim,_,(ok,_)) → (dom ikim,{ok}),
  103         mkPigU((ik,_),_,(ok,_)) → ({ik},{ok}),
  103         mkSnkU((ik,_),_) → ({ik},{})
  103     end
  103     post let (ikis,okis)=identify_KIs(u) in ikis≠{}∧okis≠{} end
```

8.2 Language Semantics s69

We shall follow the semantics suggested in Sect. 7.

104.
105.
106.
107.
108.

8.3 Language Proof Rules s70

109.
110.
111.
112.
113.

MORE TO COME

9

Photos of Pipeline Units and Diagrams of Pipeline Systems

s71

Fig. 9.1. Pipes

s72
When combining joins and forks we can construct sitches. Figure 9.4 on page 33 shows some actual s73
switches. s74

Figure 9.5 on page 34 diagrams a generic switch. s75

s76
s77

[0] See http://en.wikipedia.org/wiki/Nabucco_Pipeline

Fig. 9.2. Valves

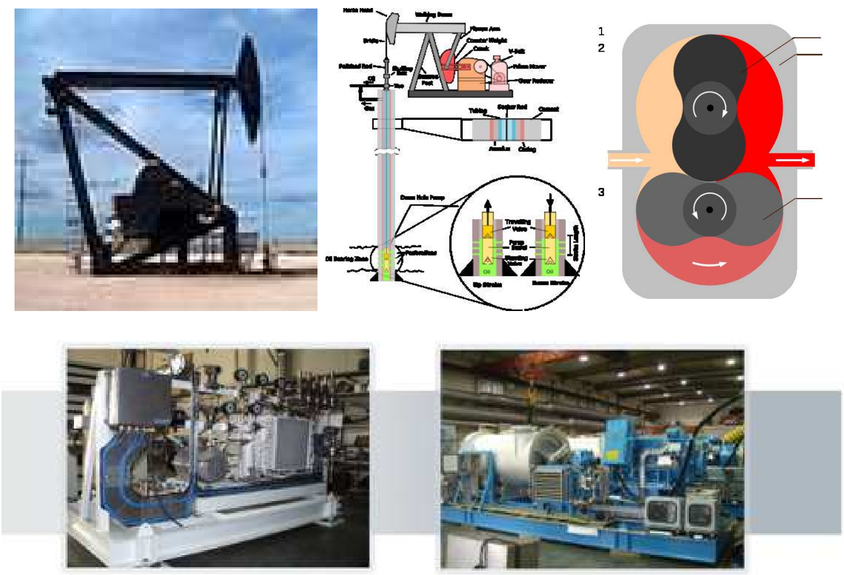

Fig. 9.3. Oil Pumps and Gas Compressors

Fig. 9.4. Oil and Gas Switches

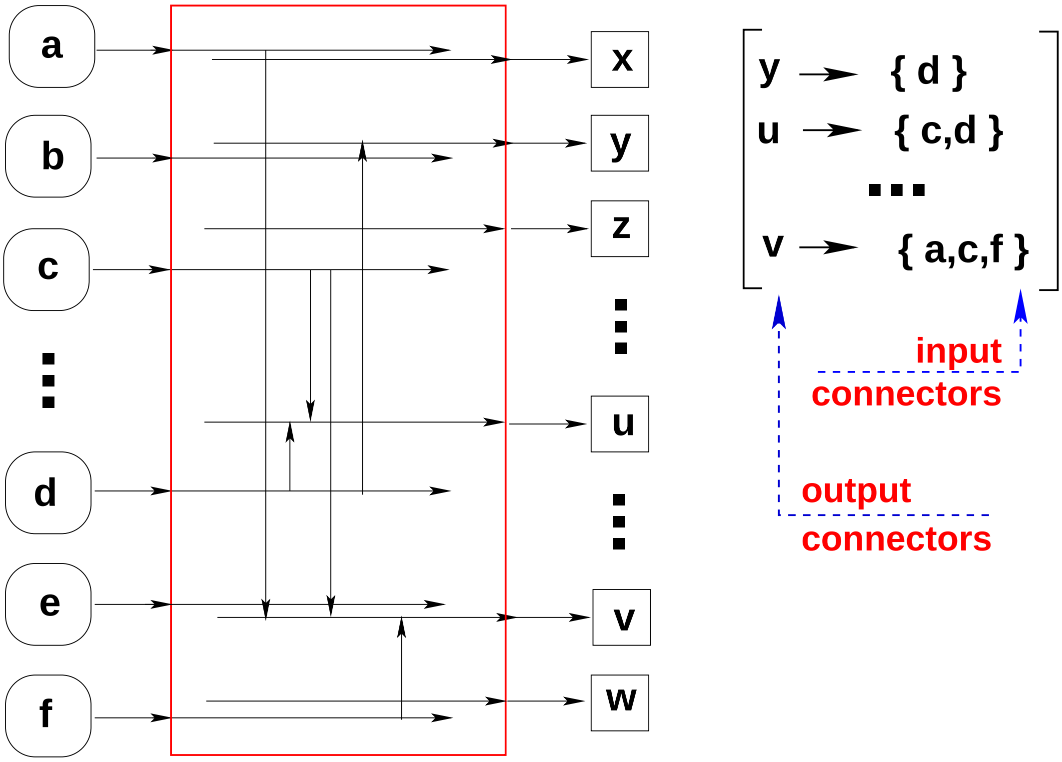

Fig. 9.5. A Switch Diagram

Fig. 9.6. To be treated in a later version of this report: Pig Launcher, Receiver and New and Old Pigs

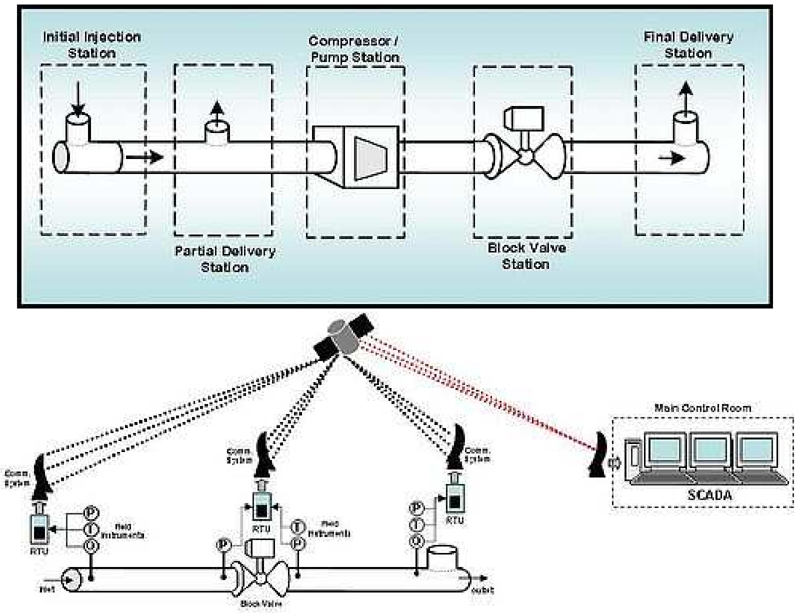

Fig. 9.7. Pipeline Diagrams