Plan B: Put in Boxes the Network Resources

Francisco J. Ballesteros

Laboratorio de Sistemas
nemo@lsub.org
http://lsub.org/who/nemo

Roadmap

- Motivation
- Problems
- Ideas
- Plan B
- Resources as boxes
- Name spaces & binding
- Problems & Lessons

Motivation: Our Computing environments

Heterogeneity

- Hardware
- Software

Dynamicity

- Of devices
- Of services

Problem: Hardware heterogeneity

Means

- Different binaries
- Different data formats
- Different devices

Problem: Software heterogeneity

Means

- Different binaries
- Different services
- Different interfaces

Is heterogeneity a problem?

It happen with devices (~ 70s)

UNIX did it well

- Device files
- Portable interfaces

Dynamic devices

They are static (mostly)

But...

- they are switched on and off
- They get connected and unreachable
- Which machine has which devices?

Dynamic services

The problems of devices

Plus...

- Interface changes
- Mutual dependencies
- Who starts which service?
- Should I start a new instance?

New services

We go pervasive:

- New services
  - power switching
  - in-door location
  - new I/O devices
- Are they input or output?

Even more problems

Environment automation

- The system makes choices
- Applications become customizable
- Where are the user preferences?
- How to automate in a different way?
- What about protection?

Ideas for heterogeneity

A common language

- UNIX, The Web, Plan 9, ...

A common interface

- UNIX, The Web, Plan 9, ...

A common set of tools

Example: What would we want?

Copy this file to the printer...

- discover which printers
- select the location for the file
- choose a compatible pair
- ask the FS to copy the file.

  cp/a/file/name /b/printer

Current status (resource selection)

Existing operating systems are not helping much their users to select which resources to use.

- Pick a file server
- Pick a printer queue
- Convert the file
- Read it, just to
- Write it back to the printer

Same for other resources...

Ideas for dynamism

- Do what we do in the real world:
  - Say what we want
  - See what we have
  - Choose what we use

Example: What would we want?

Let's navigate...

- Select the web browser application
- Pick up a mouse
- Redirect the browser's mouse input
- Pick up another
- Keep on navigating

Current status (resource redirection)

Existing operating systems are not helping much their users to change their I/O devices.

- Do a login
- Run one web browser binary
- Pick up the terminal's mouse
- Don't move!

Ideas for new services

Going pervasive

- Keep on using the common language
- Provide the new services
- Add new tools (if needed!)

What do we want?

```
while(/bin/true) {
 if (grep online /who/nemo/status)
   echo on >/n/x10/power:136light
 sleep 1
}
```

Current status (new services)

Existing operating systems are not helping much their users to access new services

- Learn which middleware to use
- Learn which interface is for X10's
- Learn which interface is for RFIDs
- Pray to find two compatible ones
- Or resort to writing yet more middleware
- Program your application

Plan B

Built for changing distributed environments

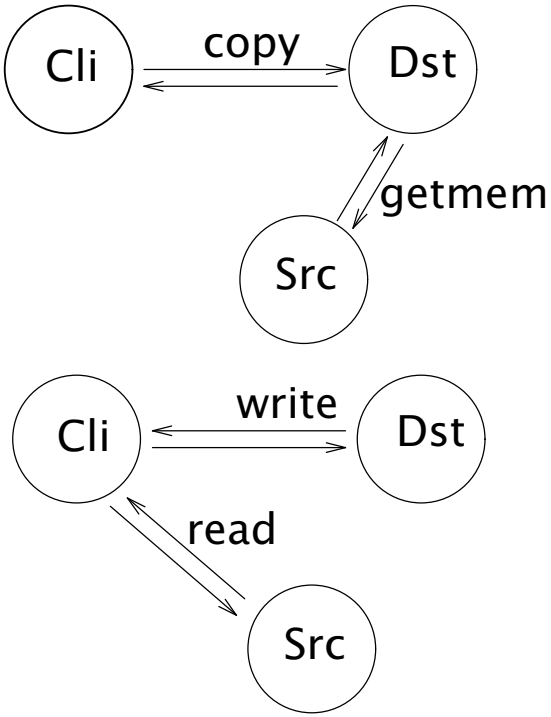
- All resources are remote
- A single abstraction: just one interface
- No binding (`open` banned).
- Single names for multiple instances
- Dynamic mounting for network resources
- Per application mount table (customizable).

Plan B: The box abstraction

A data container

- may contain other containers
- operations:
  - list /a/box
  - make /a/box
  - delete /a/box
  - copy /a/box /to/another
  - info /a/box
  - chinfo /a/box metadata

To copy or not to

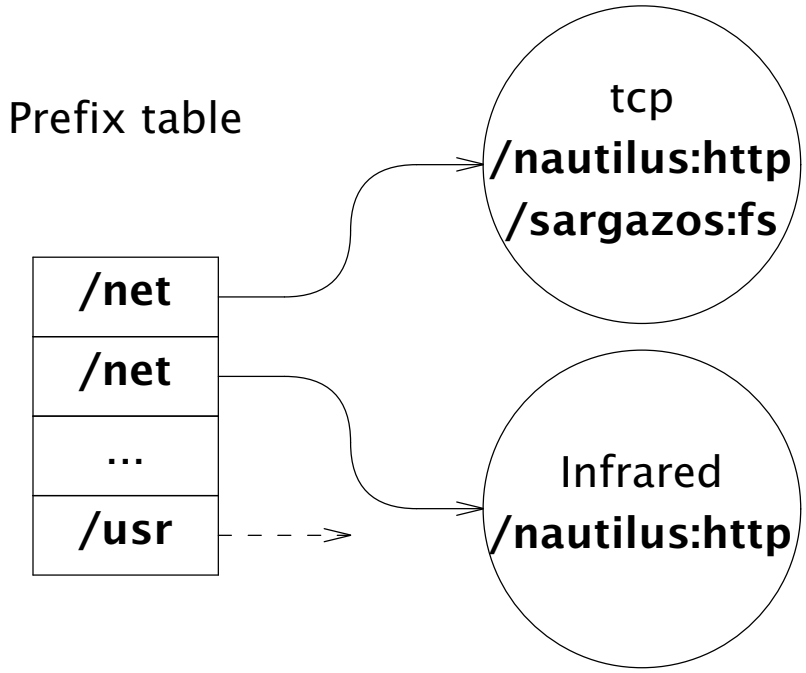


Name spaces

Tool to detect resources and adapt.

- One per application.
- Ordered prefix table.
- Order fixed at programming time.
- Dynamic (automatic) mounts.

Example: Names for networks



Name spaces (cntd.)

Constraints

- Used to narrow resolutions.
- Resolved once per system call (no binding!).
- Just syntax: users give meaning.
- Example:

```
: cp slides.mf!N% /b/printer!N%
: cp  slides.mf!Nether /b/printer
```

Box constraints

Constraints to find pairs of resources

- Boxes have properties
- System calls have constraints
- Plan B tries to unify

Box constraints

More examples

```
: import any /usr/nemo!Dgsyc /usr/nemo
: import -b any /usr/nemo /usr/nemo
: ns
...
  /usr/nemo: nautilus!fs!/usr/nemo
  /usr/nemo: aquamar!fs!/usr/nemo
: cp /usr/nemo/doc/slides.mf /b/printer
```

Box constraint unification

```
Unify(set1[1-n], set2[1-m]) {
  if for i = 1 to max(n,m)
    unifyval(set1[i], set2[i])!=fail
  then
    return
      set of unifyval(set1[i], set2[i])
  else
    return fail
}
```

Another example

`make("/b/proc/lb!A%", "/bin/lb")`

- Creates /b/proc/lb
- from /bin/ls
- such that Arch is the same.

So, did we solve...?

- Heterogeneity
- Adaptability
- Automation

Heterogeneity

We use the same interface

- I/O using strings (mostly)
- Rely on conventions

Adapting to changes

`import /b/proc any /proc proc!p98 b`

- Customize the name space for constraints
- Announce resources
- auto-mount those that unify
- Resolve names each time.

Automating things

Boxes have types (“T” constraint)

- Ta ≡ Tb for constraints to unify
  - Ta = Tb, go ahead.
  - Ta ≠ Tb, lookup the converter set
    - Entry found: convert and run.
    - Not found: unable to copy

Issues

Garbage collection

- Lease permission bit
- Remove-on-die permission bit

Considering that

- Not many things will stay orfand
- Those that will are not a problem.

Issues

Consistency: Best effort

- Replicated file systems
  - You might switch your home!
  - You might switch your binaries!
- Think what you import
  - Don’t say things are the same if they are not.

Issues

FS replication

- User-level tools
  - Tra
  - Replica
- The FS may help:
  - Report changed files

Problems

- Volatile binding
- Redirections during system calls
  - Letting the user know
  - Tiny application base

Problems

Volatile binding

- No sessions kept: price to adapt.
- Time penalty:
  - NS resolution (table search)
  - Server probing (depends on server)

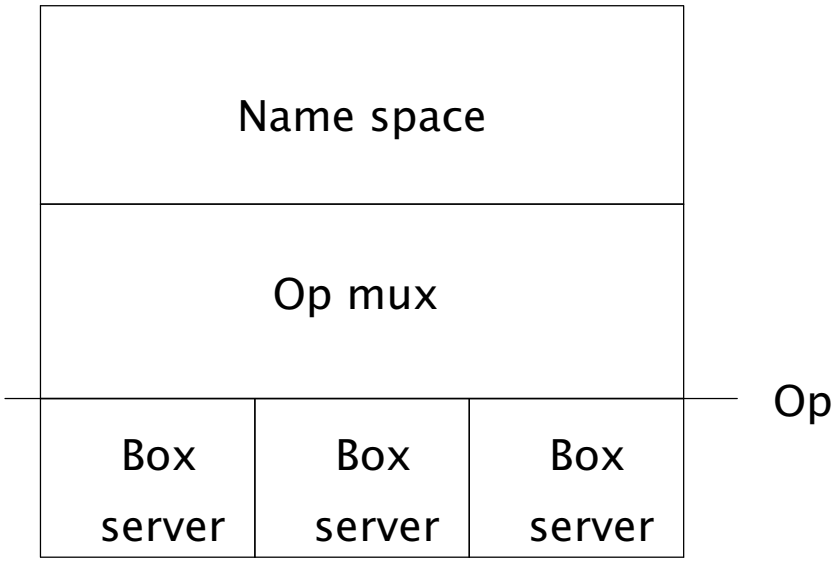
Problems

Redirections while calling

- The call is aborted.
- Retry it or not?
  - Depends on the call.
- Notify or not?
  - Exception mechanism

System architecture

A box multiplexor:



Box server interface

- list /a/box
- make /a/box
- delete /a/box
- get /a/box
- put /a/box data
- info /a/box
- chinfo /a/box metadata

Implementation strategy

Learn from vChoices, 2K, Off++, ...

- Start with a hosted implementation
- Proceed to native

Hosted implementation

Pros:

- Complete dev environment at t=0
- Fast compile-test-debug cycle
- Easy to modify
- All native services available
  - Proceed adding (virtual) replacements

Hosted implementation

Cons

- Not too fast (but acceptable!)
- Abstraction inversion
- Code thrown away when going native
  - But: Plan one to throw away [Pike, TPOP]

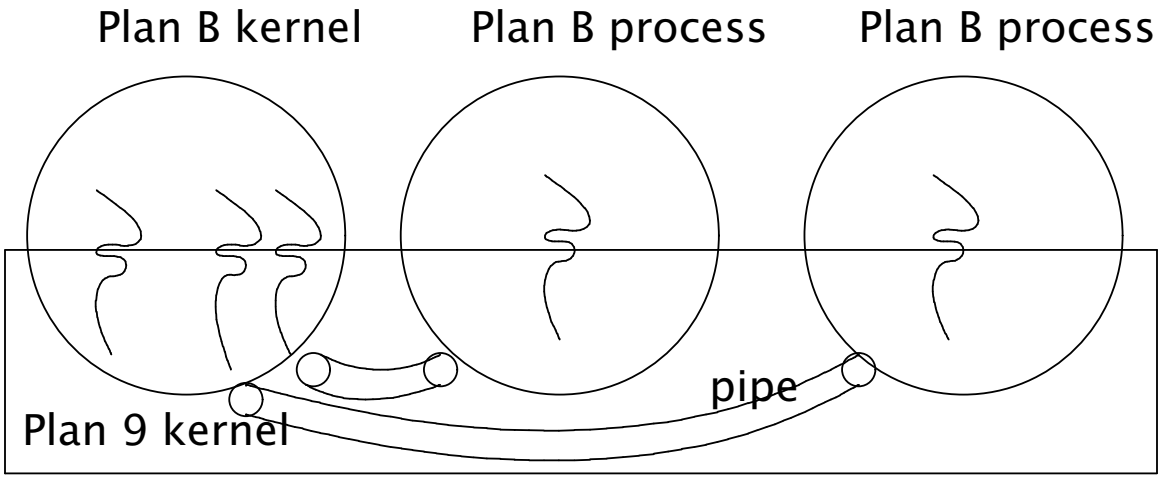
Hosted implementation

Successful experience

- Plan B 1st and 2nd editions.
- Small application bases
- Easy to use
- Experience gained quickly
  - The design modified accordingly

Hosted implementation

Uses RPCs + split process model



Native implementation

Alternatives:

- Replacing the machine dependent code.
- Modifying an existing system
  - We took this for 3rd edition Plan B.

Status

We are quite happy with the system

- Hosted version available (2nd ed.)
- Native version on production (3rd ed.)
- Now porting more applications
- Plan 9 code works out of the box

Lessons

- Connections are evil
- Names have power
- Things can be simple
- Unix did it well but for the network
- Plan 9 did it well but for dynamism
- Plan B is addictive

Questions?

More information via our httpd

`http://lsub.org`

or plan B's experimental httpd

`http://b.lsub.org`

...we call our logo "The fish"