inferno Styx protocol

An architecture for distribution
Ubiquitous, pervasive Computing

Hardware advances are making the pervasive vision true: computers are everywhere

- In the home
- At work
- On the move

However, software developments have not kept pace with the potential that new hardware offers

In particular, we still find it difficult to create reliable distributed software systems that give us access to our computing resources from anywhere
**Ubiquity** – Something that either is, or appears to be, everywhere at the same time

The software challenge is to make the resources (computers, data and services) in the network available from everywhere in the network, independent of hardware architecture, network type, operating system or programming language.

The problem of distributing the interfaces to resources is not new; computer scientists have been working on mechanisms for decades.

Although linguistics has a part to play, the fundamental solution is to be found in **PROTOCOL**
Protocol – The way computers communicate

Protocols are both a boon and a burden: they facilitate communication but are resistant to change.

Each type of resource in today’s networks are typically accessed by their own specialised protocols, based on RPC.

RPC has formed the basis of many high profile systems:
  • Xeroc Parc’s RPC
  • Sun’s RPC
  • Corba
  • JXTA
  • SOAP

And yet these RPC mechanisms have failed to make the construction of distributed systems straightforward.
Protocol – RPC – remote procedure calls

A typical RPC protocol might have a message structure like this:

<table>
<thead>
<tr>
<th>Size</th>
<th>Op code</th>
<th>Data ...</th>
<th>CRC</th>
</tr>
</thead>
</table>

The Op codes refer to the operations that can be performed on the remote resource. For example a digital camera may have the following op codes:

- 0x10 – snap
- 0x11 – zoom
- 0x12 – get image
- 0x13 – delete image

Over time, new versions of the protocol are created as the camera acquires features:

- 0x14 – insert date & time
Protocol – they are hard to change

Changes in protocol are hard to manage; backward compatibility is not easy to ensure (and prove)

Some protocol designers try to solve the problem by introducing a special op code:
- 0x15 – user defined

The data portion of the 0x15 message now contains the semantics of the message.

<table>
<thead>
<tr>
<th>Size</th>
<th>0x15</th>
<th>Data (op code, real data)</th>
<th>CRC</th>
</tr>
</thead>
</table>

Of course such a scheme is degenerative to a simpler protocol with only two messages:
- send (get)
- receive (put)
Protocol

The degenerative protocol has the advantage that the semantics of the resource are now no longer part of the communication protocol, however there are some downsides:

- The developer still has to deal with protocol issues, albeit at a higher level
- There is no structure provided to help with the composition of resources

There is clearly a balance to be struck between the general and the specific protocol. Ideally we want the invariability of the the former but with some of the structure evident in the latter.

The key is to represent all resources as the same type of object and for this object to have a small but useful set of properties. Researchers at Bell Labs chose to push to the limit a fruitful idea: ‘to present resources as files in a hierarchical name space’
Presentation of resources as Files

Representing all resource interfaces as files has several advantages:

• File systems have simple and well understood interfaces across a wide variety of operating systems
• Interfaces to files consist of a small set of well-defined operations such as open, read and write
• Reliance on file systems reduces the amount of interface code and keeps the system small, reliable and highly portable
• Naming conventions for files are well known, uniform and easily understood
• Access rights and permissions to files are simple, yet can be used to ensure multiple levels of security
STYX – providing a common language for communication

A STYX server is an agent that provides one or more hierarchical file systems – file trees or namespaces

The server responds to requests by clients to navigate the hierarchy, and to create, remove, read and write files.

A client transmits requests (T-messages) to a server which subsequently returns replies (R-messages). The full set of messages are given below:

- Tversion, Rversion
- Tflush, Rflush
- Twalk, Rwalk
- Tcreate, Rcreate
- Twrite, Rwrite
- Tremove, Rremove
- Twstat, Rwstat
- Tauth, Rauth
- Tattach, Rattach
- Topen, Ropen
- Tread, Rread
- Tclunk, Rclunk
- Tstat, Rstat
- Rerror

The protocol is small and is suitable for implementation on very small devices.
Device Interface Example

A device interface exports a namespace which is used to interact with the device.

- **ctl**: Control messages are written to this file
- **status**: Can be read to provide status information
- **images/**: Directory containing the images on the camera

These synthetic files may be accessed using standard file operations and commands.

- `echo snap > ctl`
  - Take a photo
- `echo zoom 250 > ctl`
  - Set zoom to 2.5 x magnification
- `cat status`
  - Read the status
- `ls -l images`
  - Get a list of images on the camera
- `cp images/* /pics`
  - Copy all camera images to /pics
- `rm images/0001.jpg`
  - Delete image 0001.jpg from the camera
Device Interface Example

A graphical user interface for the device simply needs to interact with the same files.
Exporting and Importing Namespaces

Each device on the network has its own local namespace. A device may export all or part of its local namespace. Another device may then import this namespace into its own local namespace. Namespaces may be built up from multiple imported local or remote namespaces. Once a namespace has been composed, the user may access each resource without needing to know its location, whether on a local or remote namespace.

As far as `cp` knows, it is just copying a local file.
DEMOS – STYX-on-a-chip

Aim to provide a coherent software architecture to support distributed application development across networks of small, resource constrained devices

We observe that architectures used in networks of larger devices don’t scale down

STYX can operate in small devices and has no difficulty scaling up

STYX is small enough that we are able to consider a hardware implementation that could become a standard component in a micro operating system