Versioning, Extensibility & Postel’s Law

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Goals

- Explore the design of data formats for systems that evolve
- Explore the consequences of connecting systems with nodes that disagree on the rules for communication
Evolving Systems, Compatibility & Partial Understanding
Terminology

- Systems evolve resulting in different versions of
  - Software
  - Data formats & protocols used by that software

- Some systems are designed from the start to be extensible
  - i.e. there is an architected approach to versioning

- Backward compatible: new versions accept old stuff

- Forward compatible: old versions accept new stuff!!!
Compatibility is a matter of degree

- **Full compatibility:**
  - Hard to define in detail
  - Spirit is: things work *right* when data shared across versions
  - *Right?* Details may still vary according to needs of app

- **Acceptable compatibility**
  - The system does something reasonable with content that received from other versions of software
  - There may be compromises (formatting, reduced function, etc.)
  - Let’s explore the notion of “doing something reasonable”
Data formats & partial understanding

- When some or all of the data is encoded and interpreted the same way by multiple versions of the specification, inter-operation is easy...we have full compatibility

- The key to flexible inter-operation is partial understanding

- *Ideally, this is achieved by specifying in early versions of the specifications how illegal content will be interpreted!*

- Huh? Is it legal or not?
  - Well, it’s in between...
  - Should be *rejected* by conformance checkers
  - ...but implementations of early versions will accept it and follow the rules
  - ...this is in the spirit of Postel’s Law
Data Format Syntax & Versioning
Challenges

- New syntax for new features
- Old features removed
- New rules for old features
- Changing the meaning of old syntax

These changes do happen...but they’re dangerous!
Challenges

- New syntax for new features
- Old features removed
- *New rules for old features*
- *Changing the meaning of old syntax*

You need something like a version id in the format to catch these!!
Problems & groundrules

- Nodes disagree about rules for interpreting messages
- We want systems to keep running *when it’s safe*
- *Certain results will not be acceptable* – we must guarantee they are never accepted
- *Not all problems are equally serious* – we’ve already discussed *partial understanding*
Unique syntax

<document>
  <order>
    ...lots of stuff here...
  </order>
</document>

<document>
  <order>
    ...lots of stuff here...
  </order>
  <comment>
    Verions 2 doesn't understand this.
  </comment>
</document>

If this is here, it always means the same thing...you either understand it or you don’t!
Unique syntax

Le’ts assume that in V1 0 means approved

For some reason in V2, the spec is changed to work the opposite way

This is dangerous! Applications can’t tell by looking whether they do or don’t understand!
Version numbers can help with this

By looking at the version number, we can tell whether we know which rules to apply.
Version numbers can help with this

By looking at the version number, we can tell whether we know which rules to apply.

Problems with Version Numbers

• Would you want to put a C version on each source file
• How would you figure out the version number?
In-band vs. in the specification controls

- **In-band controls tell the receiver what can be ignored:**
  - `<cacheThis soap:mustUnderstand="false">`
  - `<currency name="pesos" soap:mustUnderstand="true">`

- **Tolerance for extensions can be set out in the specification**
  - Unexpected HTML tags:
    - Ignored for rendering
    - Added to the DOM
    - Scriptable
    - Styleable
Postel’s Law
(the Robustness Principle)
"Be liberal in what you accept, and conservative in what you send".
Why Postel’s Law?

- Avoid brittle systems: keep running in the face of small problems
- Allow for innovation
  - We’ll discuss this later
- Hard/impossible to keep large systems in sync
  - Internet-scale systems work across organizations
- Allow for competition
  - TCP/IP re-implemented in each OS
  - Browsers: Firefox, Safari, Chrome, IE, Opera, etc.
  - Web servers: Apache, IIS
Postel’s Law is controversial

- Many in the Internet community deeply believe Postel’s law key to robustness of the Internet
  - The IETF has it as a motto
- When applied carefully at both ends, results are typically good
- You have to watch not to liberally accept something dangerous!
- When not applied carefully, big trouble results…
Problems with Postel’s Law

- HTML browsers were very *liberal* in accepting almost anything
- *So servers stopped following the rules*
- There’s a ton of buggy content on the Internet…
- …and since browsers accept it, people expect it to keep working!
- The HTML5 spec is 5x bigger than it needs to be – to *standardize* handling of buggy content
- *The servers were not conservative* in what they sent!
Note that Postel’s law says little about evolving specifications…it’s mainly to avoid brittleness in reimplementing the same specification, but…
…Postel’s law points a direction for building systems that evolve
Naming
&
Decentralized Extensibility
What if we want lots of people to add features?

- Specifications will be modular
- We need a way to recognize the syntax for each feature
- *Who gets to invent new syntax?*
- *How can we keep two organizations from inventing the same syntax?*

Problem: as we saw when we studied naming...

...globally unique names tend to be ugly and inconvenient
The HTML Distributed Extensibility Debate

- The W3C spent years moving HTML to XML (XHMTL)
- A big reason was so anyone could invent new tag names *without collision*...XML has *namespace*
- But... many users rebelled against the inconvenience
- HTML5 can only be updated by a central committee...
- ...but the tag names are short and convenient
- *This controversy almost derailed HTML evolution*