Call Processing Language

Jonathan Lennox
Bell Laboratories / Columbia University
lennox@bell-labs.com

IETF IPtel Working Group
August 24, 1998

Architecture

- CPL scripts everywhere
  - Network servers
  - End systems
- Transaction-based scripts: persist from initial request to final response
- Script transport separate from script design

  Remove close coupling with SIP Register message (though that could still transport scripts).

Feature interactions — 1

A feature interaction is a condition when having several features specified creates a conflict between the two.

- Feature-to-feature
  In traditional telephony, several features in a single server can specify conflicting behavior for a given situation.
  - CPL behavior based on conditions (“call arrives while line is busy”), not on named features (“call waiting” vs. “call forward on busy”).
  - Interaction is thus not a problem in a CPL environment.

Feature interactions — 2

- Script-to-script in a single server
  If several scripts specify the behavior for a call, it can be unclear which one to follow.
  - In a CPL environment, a user specifies only a single script at a time.
  - Administrative scripts run after user scripts, intercepting proxy or redirect decisions.
- Server-to-server
  Several separate servers can implement features which conflict.
  - Some interactions are the signaling protocol's responsibility:
    Forwarding loops
  - Some are unavoidable:
    Outgoing call screening (server A) vs. call forwarding (server B)

Signalling server $\Longleftrightarrow$ language environment interface

- Independent of implementation language
- Possibly specific to signalling protocol
- For SIP, could be similar to CGI-bin
- For single request/response (redirect server), carries over easily from CGI
- For multiple requests/responses in a transaction, must be more complex — event-driven.
- Open: How to handle this case?

User-created language

- What end systems send to servers
- An actual language, as motivated by the draft
- Idea: XML-based
  - Service Creation Environments in IN use decision trees. Thus, tree structure is sufficient to describe services.
  - XML is an established syntax with freely available parsers.
  - It is easy to parse and write for both humans and computers.

XML-based language: example

approximate syntax only:

```
<call>
  <proxy dest="sip:lennox@phone.cs.columbia.edu" timeout="8s">
    <busy>
      <redirect dest="sip:lennox@voicemail.cs.columbia.edu"/>
    </busy>
    <timeout>
      <condition from="hgs@*cs.columbia.edu">
        <match> <gateway dest="phone:+19175551212"/> </match>
        <nomatch>
          <redirect dest="sip:lennox@voicemail.cs.columbia.edu"/>
        </nomatch>
      </condition>
    </timeout>
  </proxy>
</call>
```

Open issues: features

- Choosing among multiple responses

  How long do we wait for all targets to respond?

- Granularity of primitives
  - High-level features: Queueing, call distribution combine a number of actions:
    - inter-transaction notification
    - global state
    - provisional responses
    - complex timers
  - Should these be primitive features, or should they be creatable?

Open issues: design

- How “Call”-specific should this be?
- “Communications Processing Language”?
- A language could apply to:
  - fax
  - e-mail
  - presence

  Is this a good idea?