A Policy Architecture for Enhancing and Controlling Features

Stephan Reiff-Marganiec

Kenneth J Turner

University of Stirling

UNIVERSITY OF STIRLING

Context of Research

- ACCENT Project
  - Advanced Call Control Enhancing Network Technologies
  - 2001-2004
  - EPSRC
  - Mitel
- Goal:
  - define a comprehensive and practical policy language for call control

Outline

- Motivation & Background
  - Features & Policies
- A Policy Architecture
- Policy Conflict

- Defining & Deploying Policies
- Enforcing Policies

- Conclusions

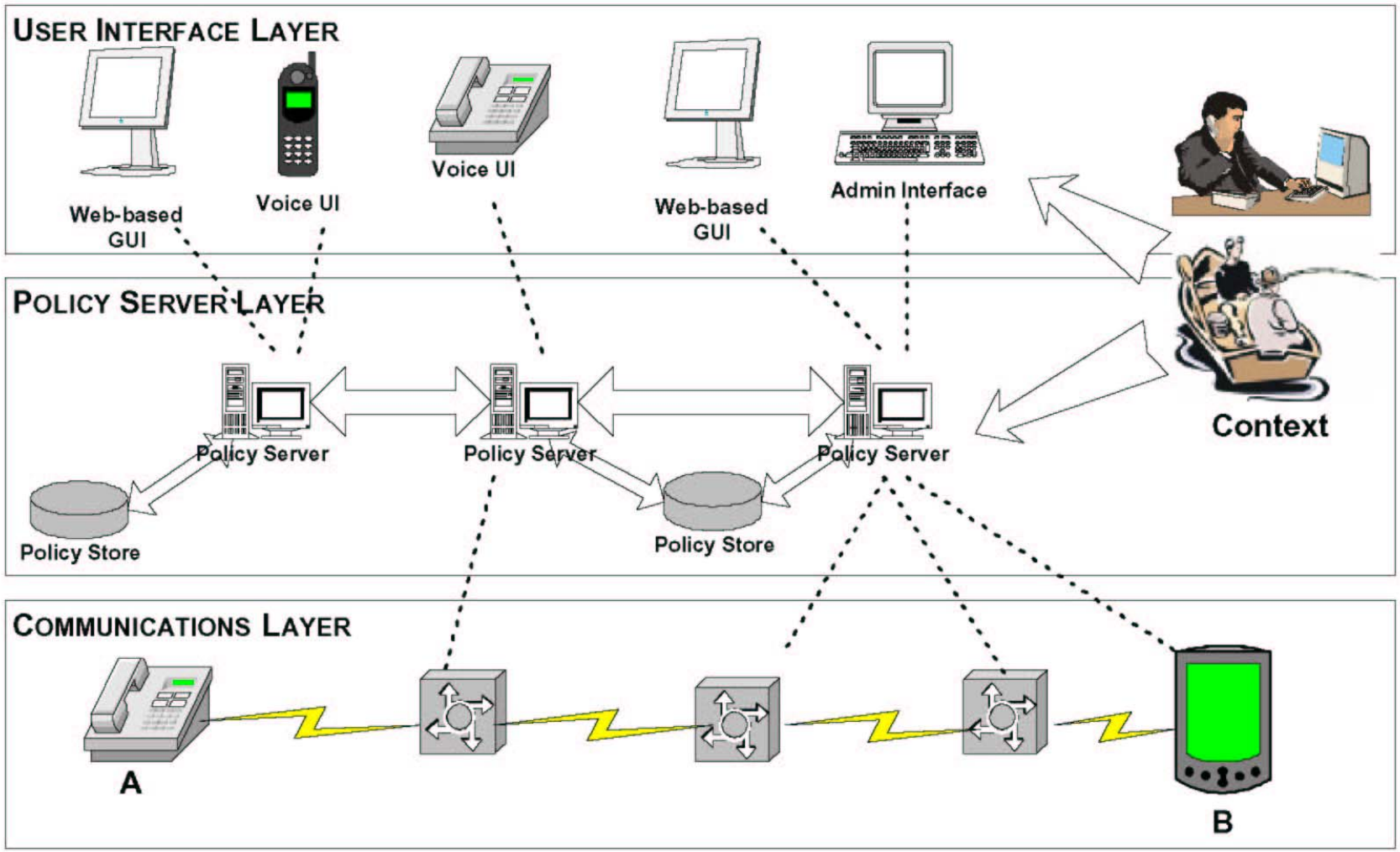
Motivation

- Technology changes
  - merging of communications technologies
    - mobility, ad-hoc networks, multiple devices, …
- User requirements
  - users are “always on”
    - but might not always want to be disturbed
  - Services must provide availibility control
  - Availibility depends on context
  - End users should specify the behaviour they wish
    - simple and intuitive design, suitable for lay users
  - **End users must be central**

Features and Policies

- Features
  - from service providers
  - minimal end-user configurability
    - CFU example
- Policies
  - “information modifying behaviour of system”
    - ODP, QoS, …
  - can be formulated by end-users
- However
  - require appropriate languages, supporting architectures and development processes

Enhanced Call Control Architecture

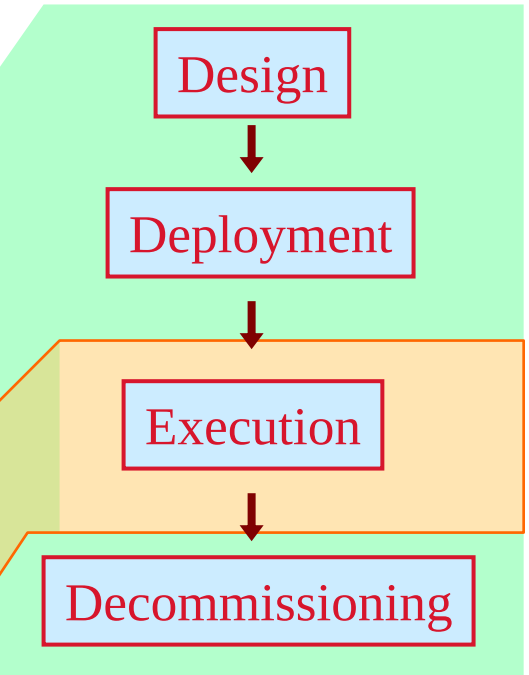


Policy Conflict: The Problem

- The FI problem re-occurs
  - Two or more policies might contradict
- Good news:
  - Policies can express user preferences
  - Rich protocols allow for negotiation
- Bad news:
  - There will be many more policies than there have been features
  - Hierarchies (e.g. enterprise and user policies)
  - Policies might be written by lay users

Handling FI and PC

- Feature Interaction and Policy Conflict must
  - be detected
  - be resolved
- requires
  - design time environments
    - that allow automatic detection,
    - and suggest concrete solutions
  - runtime environments
    - that allow automatic detection,
    - and automatic resolution



Handling FI and PC – Offline

- offline = design-time
  - static analysis detects problems
    - (FM, Testing, Design Principles)
  - resolution by redesign
  - good if details are known (intra-company, ...)
  - for policies automatic methods can be used at upload time, user then can redefine policies
  - not suitable when design details are unavailable (open market)

Handling FI and PC – Online

- online = run-time
  - dynamic analysis for detection
  - automatic resolution
    - lookup tables (early approaches)
    - domain specific, general rules
    - mutually best (negotiation)
  - two main classes, but little work
    - FMs [Cain, Marples, Reiff-Marganiec]
    - Negotiation [Velthuijsen]
  - can handle black-box features/ policies

ACCENT Policy Language

policy_rule ::=

[triggers] [conditions] actions

- triggers and actions are domain specific

policy ::=

"preference" "applicable to"
(policy_rule | policy_rule op policy_rule)

- where op is sequential, parallel, choice

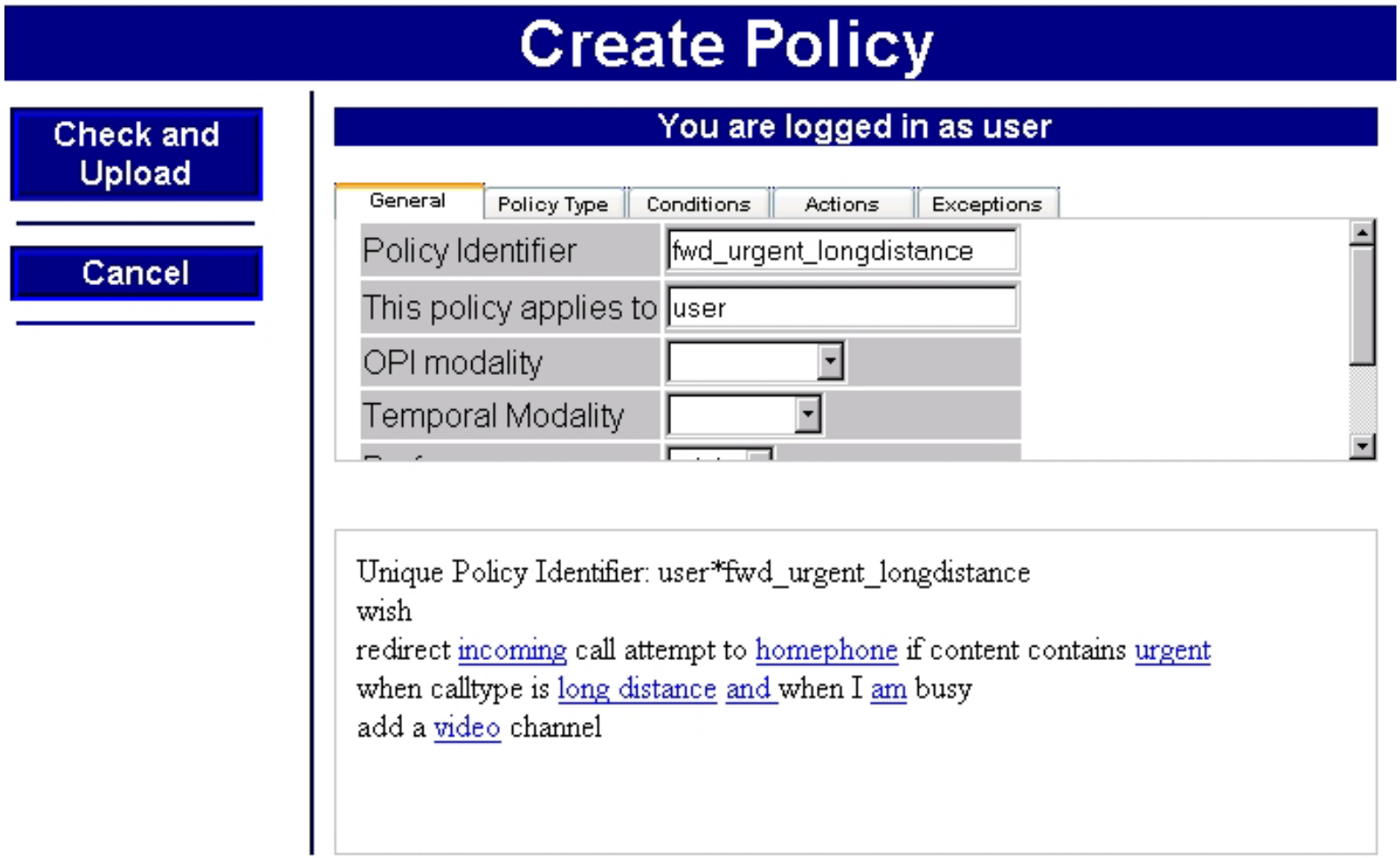
- Language defined in XML
- User has "wizard" to define policies

[Reiff-Marganiec, Turner: FORTE 2002]

Example Policies

```
<policy owner="srm@cs.stir.ac.uk" appliesTo="srm@cs.stir.ac.uk"
        id="Mary_after_1900" enabled="true">
   <policyrules><polrules><policyrule>
        <triggers>
                <trigger>incoming</trigger>
        </triggers>
        <conditions><and/><conds>
                <condition>
                        <param>caller</param>
                        <compop>eq</compop>
                        <value>Mary</value>
                </condition></conds><conds><condition>
                        <param>time</param>
                        <compop>gt</compop>
                        <value>1900</value>
                </condition></conds></conditions>
        <actions><acts>
                <action>connectto(home)</action>
        </acts></actions>
   </policyrule></polrules></policyrules></policy>
```

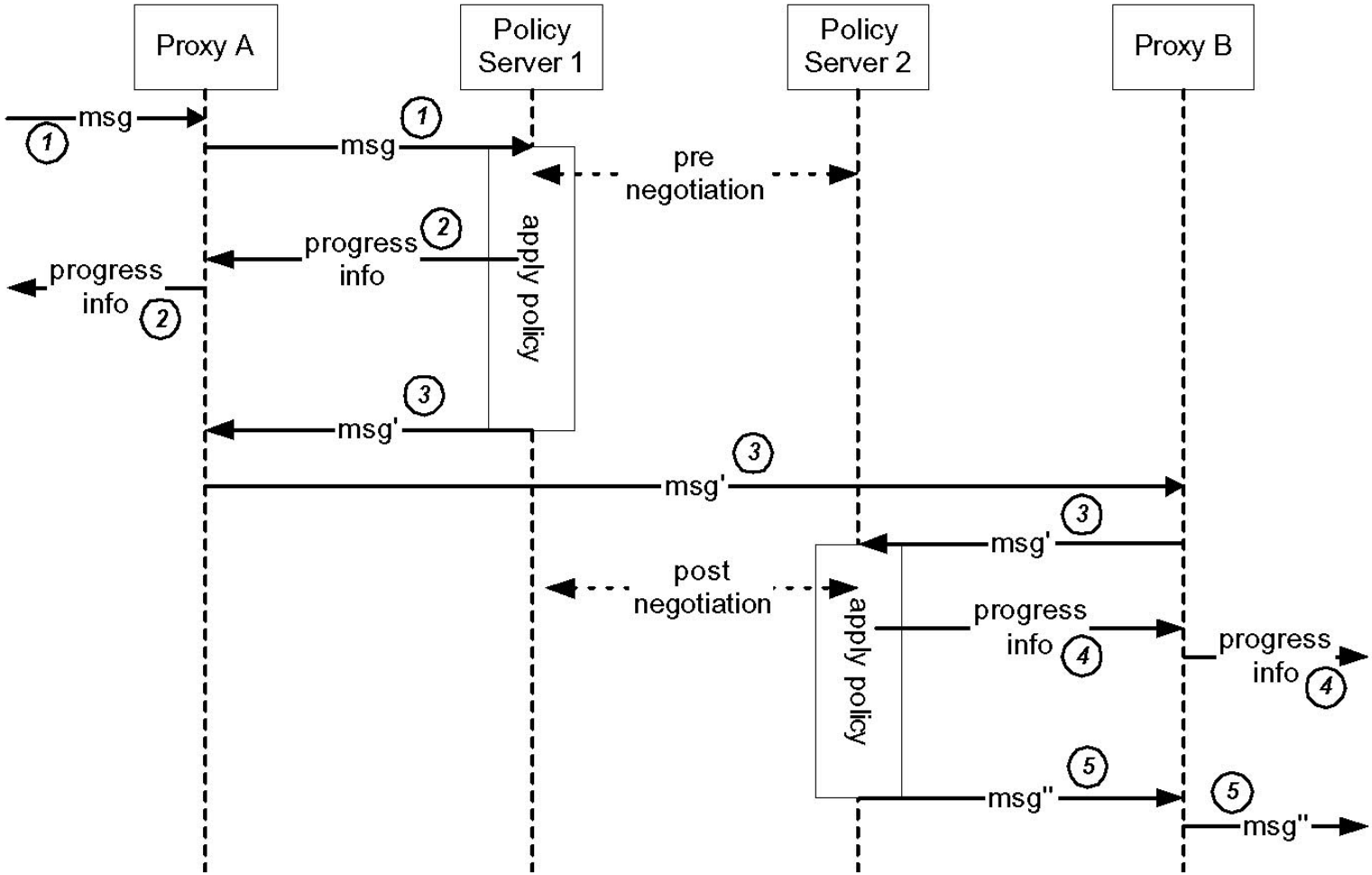
Policy Wizard



Handling Policy Conflict (1)

- Policy upload
  - check users policies for consistency
  - check users policies against known domain policies
  - suggest solutions & describe problem
  - allow user to select solution or redefine policies
- Policy Enforcement …
  - combining ideas of FI online approaches
  - agent architectures

Handling Policy Conflict (2)



static interactions: an example

enterprise.com has existing policy:

- all calls during working hour should be answered by a person within 5 rings.

me@enterprise.com defines new policies:

- if I don't answer calls within 3 rings forward them to my voicemail if it is not my boss.
- when visitors arrive at reception notify my secretary

check policies defined by user ✓

check user vs. domain policies ✗

caller might get voicemail

dynamic interactions: an example

mary@enterprise.com has policy:

- I prefer to speak to John if Paul is busy.

paul@elsewhere.com has policy:

- I expect that my calls are redirected to Joanne when I am busy.

•Mary rings Paul
•Paul is busy

Mary rings Paul; Paul is busy

conflict: forward to Joanne or John??

✓ Joanne: using preference

? could also negotiate ...

Conclusions

- Call control can be achieved with policies
- High-level user goals
- Both, online and offline methods required to handle conflict

- User is central
- User must have control

any questions?

more details:

- {srm,kjt}@cs.stir.ac.uk

- http://www.cs.stir.ac.uk/{~srm,~kjt}
- http://www.cs.stir.ac.uk/compass